GE

Form Approved
OMB No. 0704-0188

1. AGENCY USE	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1994	Technical Report, 1/1/94 -6/30/94

4. TITLE AND SUBTITLE

ADAPTIVE DECISION MAKING AND COORDINATION IN VARIABLE STRUCTURE ORGANIZATIONS

5. FUNDING NUMBERS

N00014-93-1-0912

6. AUTHOR(S)

Alexander H. Levis

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Center of Excellence in Command, Control, Communications and Intelligence
George Mason University
Fairfax, Virginia 22030

8. PERFORMING ORGANIZATION REPORT NUMBER

GMU/C3I-153-IR

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
Arlington, Virginia 22217-5000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Progress in research on coordination in distributed decision making organizations with variable structure is reported. The problem of consistency and completeness of the set of decision rules used by an organization is addressed by modeling the rule base by a Colored Petri Net and then analyzing the static and dynamic behavior of the net. The design problem is addressed by (a) focusing on algorithms that relate structural properties of the Petri Net model to behavioral characteristics; and (b) by incorporating design requirements in the Lattice algorithm.

14. SUBJECT TERMS

Decision making; Organization theory; Coordination; Colored Petri Nets; Rule based systems

15. NUMBER OF PAGES

37

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

CENTER OF EXCELLENCE IN
COMMAND, CONTROL, COMMUNICATIONS AND INTELLIGENCE

GEORGE MASON UNIVERSITY
Fairfax, Virginia 22030

SEMIANNUAL TECHNICAL REPORT

for the period

1 January 1994 - 30 June 1994

for

ADAPTIVE DECISION MAKING AND COORDINATION IN VARIABLE STRUCTURE ORGANIZATIONS

Grant Number N00014-93-1-0912

Submitted to
Dr. W. S. Vaughan, Jr. (3 copies)
Office of Naval Research
800 North Quincy Street
Arlington, Virginia 22217-5000

Submitted by:

Alexander H. Levis
Principal Investigator

Copies to:
Director, Naval Research Laboratory
Administrative Grants Office, ONR
Defense Technical Information Center

September, 1994

Report #: GMU/C3I-153-IR

1. PROGRAM OBJECTIVES

The objective of this research, as described in the proposal and the previous progress report, is the investigation of several issues related to coordination in organizations. In particular, an organization is coordinated through direct and indirect means. The direct means includes the set of decision rules that the organization members use and the commands that they issue to each other. Indirect means include the dissemination of information within the organization; for example, organization members may share information or they may inform each other as to the actions they plan to take or decisions they have made. Coordination becomes a complex issue in variable structure organizations. Not only do the decision rules and the information architecture have to work for each fixed structure, but the designer has to deal with the problem, a metaproblem, of coordinating the variability. This becomes a particularly difficult problem in organizations that exhibit substantial complexity and redundancy in their information structure. The redundancy is necessary both for robustness and for flexibility and reconfigurability. In order to address these problems two main tasks were defined; they are described in the next section. In addition, some basic work in algorithms and Colored Petri Nets needs to done to develop tools and techniques for supporting the analysis and design.

2. STATEMENT OF WORK

The statement of work, as outlined in the proposal, is given below.

Task 1: Consistency and Completeness in Distributed Decision Making

Develop a methodology for analyzing and correcting the set of decision rules used by an organization with distributed decision making. The methodology is to be based on the modeling of the set of decision rules in the form of a Colored Petri Net and on the analysis of the net using S-invariant and Occurrence graphs. The ability to verify and correct the set of decision rules has direct impact on the extent of coordination needed in an actual organization and the resulting communication load.

Task 2: Variable Structures: Heuristic rules in the Lattice Algorithm Constraints

Develop a methodology for considering additional constraints in the Lattice Algorithm. Such constraints include the degrees of redundancy and complexity at the different processing nodes (to be derived from the DFS algorithm of Andreadakis), the projected response time of the organization, ands some user-specified constraints on connections between decision making units. Develop a procedure for checking the validity of such constraints and incorporate them in the Lattice Algorithm. Generalize the approach to multilevel organizational structures and to variable structures, where variable structures are obtained by folding together different fixed structures. The real focus of the task is to introduce these additional constraints as a way of containing the dimensionality problem inherent in flexibility and reducing the coordination requirements.

Design a symbolic interface for the Lattice algorithm. The interface would have the capability of interpreting natural language inputs entered by the user and will include some symbolic processing. The system will generate the interconnections matrices used as input to the Lattice algorithm. The designer would then use the various tests described in the proposal (such as DFS algorithm) to check the validity of the interconnection constraints and to make required modifications.

Task 3: Information Dissemination

Semiannual progress reports are submitted in place of annual reports. The results of this research will appear in thesis reports and in technical papers to be presented at professional meetings and published in archival journals. In addition, oral presentations will be given periodically ad arranged with ONR.

3. RESEARCH PLAN

The research plan describes the strategy for meeting the program objectives. Specifically the research plan is organized around a series of specific well-defined research tasks that are appropriate for theses at master's and Ph.D. levels. Individual students are assigned to each task under the supervision of the principle investigator. Additional staff from the C3I Center are included in the project whenever there is a specific need for their expertise.

The focus of the task 1 is the development of a methodology for analyzing and verifying the set of decision rules used by an organization with distributed decision making. The methodology is based on the modeling of the decision rules in the form of Colored Petri Net and on the analysis of the net using S-invariant properties and Occurrence graphs. The results obtained for the two analyses, when applied to a specific form of decision rules, have been presented in the previous report. During the reporting period, the results were extended to decision rules expressed in First-Order Predicate Calculus. The next task is to further extend these results for a general form of rule bases that are expressed either in Predicate Calculus or First-Order Predicate Calculus. This work is being done by A. Zaidi as part of his Ph.D. dissertation.

Task 2 has been initiated and focused on the consideration of additional constraints for small non-variable organizations (case studied by Remy, where the number of decision makers is less than 5). The consideration of degrees of redundancy and complexity at different processing nodes, the projected response time of the organization, and the consideration of user-defined constraints on connections between decision making units were investigated. The inclusion of these constraints does not introduce any drastic changes in the way the problem of generating feasible structure is approached by the Lattice Algorithm. The approach needs to be generalized to multilevel structures and variable structures. This generalization will be the focus of the effort during the next period.

A second issue identified in the previous report that affects directly the use of the Lattice algorithm in task 2 is computation of S-invariants and extension of the algorithm to the determination of deadlocks and traps in the organizational structure. The results obtained during this effort have also been used in task 1. Ms Jin has completed her Master's thesis on this subject; the technical report is in preparation.

4. STATUS REPORT

In the context of the project tasks and research plan outlined above, a number of specific research tasks have been formulated. Each research task is discussed below. Detailed results for some of the research tasks have already been presented in the previous semi-annual report.

4.1 CONSISTENCY AND COMPLETENESS IN DISTRIBUTED DECISION MAKING

Detailed results of the S-invariant and Occurrence graph analyses applied to a Colored Petri Net representation of a rule base has already been presented in the previous report. The two analyses have been shown to uncover hidden incomplete, inconsistent, circular, redundant, and subsumed cases in a set of decision rules expressed originally as statements in either Propositional Calculus (PC) or First-Order Predicate Calculus (FOPC). During this reporting time period a number of techniques, presented in the previous report, were refined and checked for any overlooked issues. The effort resulted in some slight modification (mostly additions) to these algorithms. It has been stated in the previous report that the techniques presented so far are applicable to rule bases expressed as statements in PC. This report extends the results obtained for PC systems to include rules bases in FOPC.

For the rule bases in First-Order Predicate Calculus (FOPC), the Colored Petri Net (CPN) representation of the rule base is first converted to the Associated Petri Net (APN) representation. The APN is the underlying Petri Net structure of the CPN in hand. The S-invariant and Occurrence graph analyses presented so far are now applied to the APN and results from the two analyses are applied to the APN. The results so obtained only identify the structures of the PNs that correspond to potential problematic cases; The reported cases provide the necessary conditions for the problematic cases, however, one needs to apply further analysis to prove the sufficiency. Once the potential problematic cases are reported by the first phase of the analysis, a second phase is required to check the variable assignments of the terms involved in the reported cases to identify the real problems. The following sections describe Phase II of such an analysis in detail.

4.1.1 S-Invariant Analysis for FOPC Systems

The following steps summarize the techniques used in the S-Invariant analysis of the APN corresponding to a CPN representing a rule base in FOPC;

- For a rule base in FOPC, the corresponding CPN is converted to an APN.
- Remove all the dangling places, in the APN, identified during detection of incompleteness.
- Construct the Marked Petri Net (MPN) from the APN.
- Calculate minimal support S-invariants of the MPN.
- Search calculated minimal supports for problematic cases (identified in previous reports).
- For systems in the restricted form of FOPC (4.1, below), report the problematic cases, otherwise perform Phase II of the analysis (see below).

The following restricted form of FOPC does not require any further analysis;

$$\forall x \, (\alpha \rightarrow \beta) \tag{4.1}$$

where α and β are literals composed of unary predicates only; the terms involve only one variable x.

The reported plausible problematic cases from the first phase of the analysis are grouped into the following two sets:

IR: the set of plausible inconsistent rules;
CR: the set of plausible circular rules.

Phase II

As mentioned earlier, the definition of problematic cases is more involved in FOPC than the one for rules in PC. The application of the algorithm on systems which are originally expressed in FOPC only identifies the Petri Net structures of rules that might have problems. This section presents algorithms which take these reported cases and check the variable assignments and bindings for possible erroneous instantiations of these rules. If such an assignment or binding is found, the rule is declared as problematic. The algorithm also identifies the erroneous instantiations of these rules; this information might help an expert to remove the errors by putting constraint (exceptions) on variable bindings. The algorithm presented in this section uses the following technique.

- Construction of Generalized Deduction Tree

The approach is similar to the Explanation-Based Learning (Mitchell et. al., 1986, 1990; Dejong and Mooney, 1986, 1990; Tecuci, 1992).

Algorithm for Inconsistent Rules (Case I & III)

(The set of possible inconsistent rules IR is given.)

- Construct a generalized deduction tree with the help of rules in IR; the deduction tree can be directly constructed from the CPN representation of rules in IR.
- In the generalized deduction tree, check the terms associated with predicates p and q, where the two predicates are mutually exclusive (semantically or syntactically). If the terms associated with these predicates are identical, the set of rules IR is in fact inconsistent.
- In case the set is not identified as inconsistent, calculate the variable assignment (if one exists) that will generate an inconsistent instantiation of IR.

The following example illustrates the algorithm: Consider the set of rules (IR1) that is reported as (possibly) inconsistent during the first phase of the algorithm;

R1: $\forall x \forall y [A(x, y) \wedge B(y) \rightarrow C(x, y)]$

R2: $\forall u \forall v [C(u, v) \wedge D(v, u) \rightarrow E(v, u)]$

R3: $\forall x \forall y [E(x, y) \wedge F(y) \rightarrow \neg A(y, x)]$

The CPN representation of these rules is given in Figure 1.

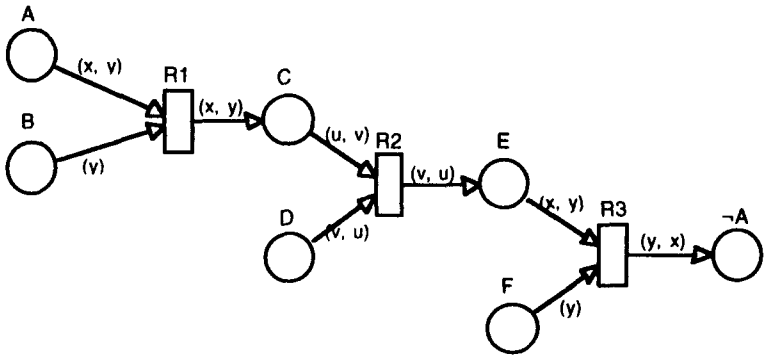

Figure 1 CPN Representation of Rules

The two steps of constructing the generalized deduction tree from the CPN in Figure 1 are illustrated in Figure 2. The first deduction tree is constructed directly from the CPN; the terms of the predicates are compared at each node in the tree in order to find the most general unification pattern of terms. In the deduction tree (Figure 2), the comparison of the terms yields the following unification patterns:

$$E(v, u) \equiv E(x, y) \qquad [v(\text{in R2}) = x(\text{in R3}),\ u(\text{in R2}) = y(\text{in R3})]$$

$$C(x, y) \equiv C(u, v) \qquad [x(\text{in R1}) = u(\text{in R2}),\ y(\text{in R1}) = v(\text{in R2})]$$

The most general unification pattern of variables will, therefore, be given as;

$$[x(\text{in R1}) = u(\text{in R2}) = y(\text{in R3}),\ y(\text{in R1}) = v(\text{in R2}) = x(\text{in R3})]$$

The terms in R2 and R3 are modified to reflect this unification; the resulting tree is the most general deduction tree and is also shown in Figure 2. Since the terms of Predicates A and ¬A are identical, the set IR1 is in fact inconsistent.

Consider now a second example with the following set of rules (IR3) that is reported as (possibly) inconsistent during the first phase of the algorithm;

R1: $\forall x\ \forall y\ [A(x, y) \wedge B(y) \rightarrow D(x)]$

R2: $\forall x\ \forall y\ [C(y) \wedge \neg A(y, x) \rightarrow E(x)]$

R3: $\forall x\ \forall y\ [D(x, y) \wedge E(y) \rightarrow F(x, y)]$

The CPN representation of these rules is given in Figure 3. The generalized deduction tree is shown in Figure 4.

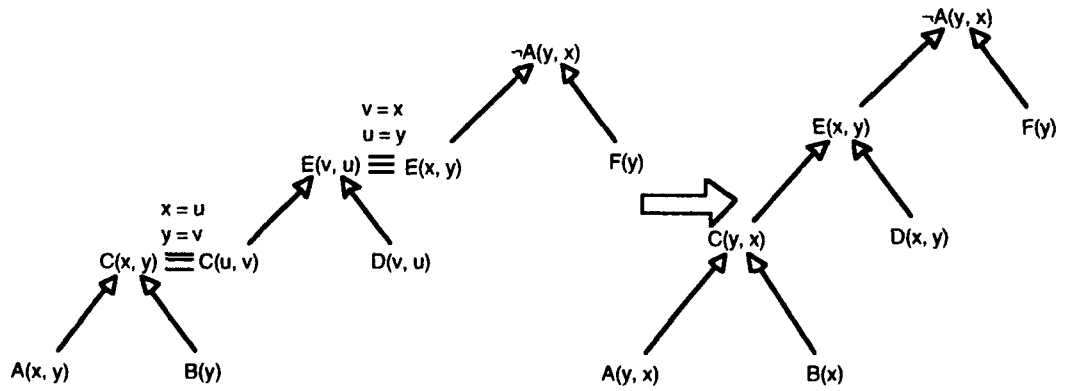

Figure 2 Construction of Generalized Deduction Tree

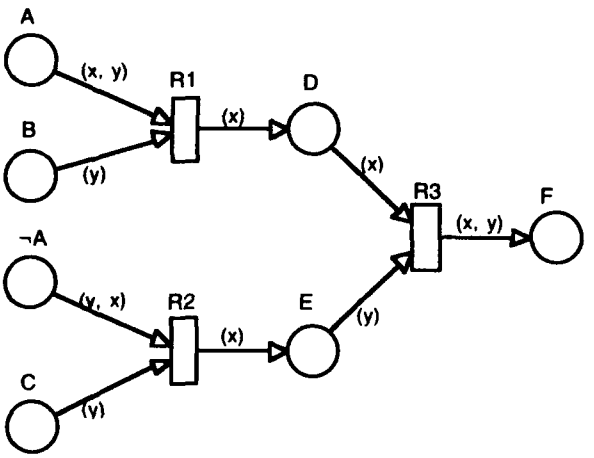

Figure 3 CPN Representation of Rules

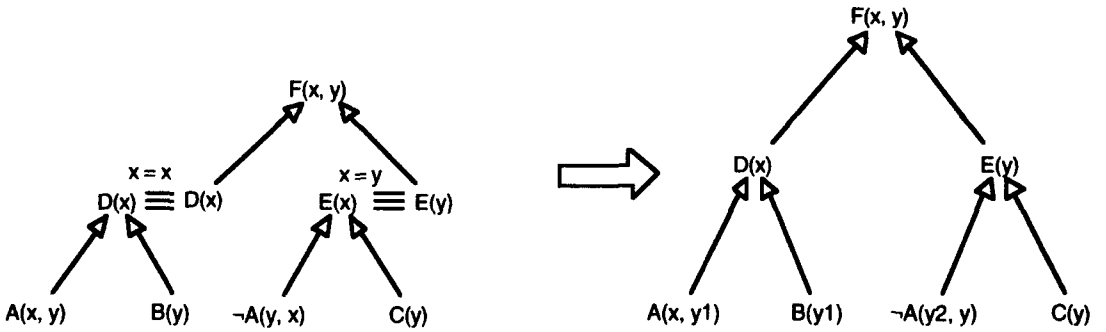

Figure 4 Construction of Generalized Deduction Tree

The set of rules is not inconsistent since the terms associated with A and ¬A are not identical; they represent the relation or property between two different set of objects (constants). However, the following instantiations of variables in R1 and R2 will introduce an inconsistency;

x (in R1) = y (in R2)
y (in R1) = x (in R2)

and/or

x (in R1) = x (in R2)
y (in R1) = y (in R2), in case the relation A is symmetrical, i.e., A(x, y) = A(y, x)

Algorithm for Circular Rules

(The set of circular rules CR is given together with the minimal support <Xi> of MPN that helped identify these rules.)

- If $\forall p, p \in <Xi>, |^{\bullet}p| = 1$ in the original CPN, then <Xi> represents a deadlock and no further analysis is required. Report the case as erroneous.
- Construct a generalized deduction tree with the help of rules in CR: In order to be consistent with the definition of a tree, the loop in CR is opened by duplicating one of

the places (selected randomly), p, in <Xi>. The deduction tree can now be directly constructed from the CPN representation of rules in CR with one node, representing p, appearing twice.

- In the generalized deduction tree, check the terms associated with the duplicated node, p. If the terms associated with these predicates are identical, the set of rules CR is declared circular.

The approach is illustrated in the following example. Note that the algorithm described above checks a single iteration of the circular rules to determine the error. There might be cases that are circular in the second, third or n^{th} iteration, but since these cases do generate new assertions, they are not considered erroneous by the algorithm.

Consider the following example: a set of rules (CR) is reported as (possibly) circular during the first phase of the algorithm.

R1: $\forall x \forall y [A(x, y) \wedge B(y) \rightarrow C(x, y)]$

R2: $\forall u \forall v [C(u, v) \wedge D(v, u) \rightarrow E(v, u)]$

R3: $\forall x \forall y [E(x, y) \wedge F(y) \rightarrow A(y, x)]$

The CPN representation of these rules is given in Figure 5. The shaded place and arc in the CPN represent the process of opening the loop so that a generalized deduction tree can be constructed. The generalized deduction tree is shown in Figure 6.

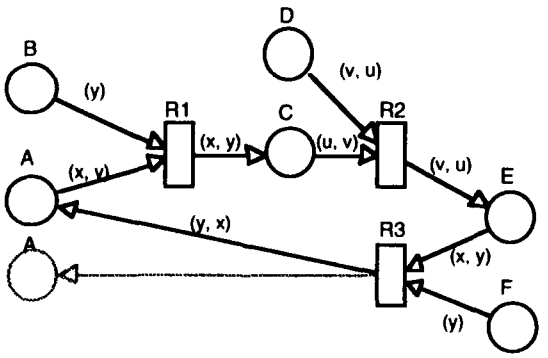

Figure 5 CPN Representation of Rules

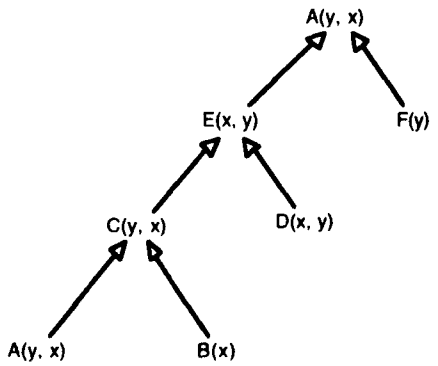

Figure 6 Generalized Deduction Tree

The terms associated with predicate A in the deduction tree indicate that the set CR in fact consists of circular rules.

4.1.2 Occurrence Graph Analysis for FOPC Systems

As was the case with S-invariant analysis, the Occurrence graph analysis presented for PC systems is applied to the APN representation of the CPN representing a rule base in FOPC. During this initial phase, the reported cases only identify the Petri Net structures of rules that might have problems. This section presents algorithms which take these reported cases and check the variable assignments and bindings for possible erroneous instantiations of these rules. If such an assignment or binding is found, the rule is declared as problematic. The algorithm also identifies the erroneous instantiations of these rules; this information might help an expert to remove the errors by putting constraint (exceptions) on variable bindings. The algorithm presented in this section uses the following two techniques;

- Construction of Generalized Deduction Tree
- Unification of Rules

The first approach has already been presented and illustrated in the previous section. The first phase of the Occurrence graph analysis results in two (or more) sets of rules for every identified redundant, subsumed, and conflicting case. The approach constructs generalized deduction trees for every such set to generate the most general expressions representing rules in these sets. The two generalized expression are compared in the second approach to establish the existence of a problem. The approach unifies the variables in the two generalized expressions and compares them for variable assignments that might lead to a problematic instantiation.

Algorithm for Redundant Rules

(Two sets of rules RR1 and RR2 are found with structures representing redundancy — rules in RR1 are redundant in the presence of rules in RR2 and vice versa.)

- Construct a generalized deduction tree for rules in RR1. Construct another generalized tree for the rules in RR2.
- From the two generalized deduction trees, extract the most general form of the rules in the two sets by collecting the leaves of the tree as premises and root as the consequent.
- Unify the variables in the terms in the two general expressions obtained. If unification results in two identical rules, declare the rules as redundant.
- In case the sets are not identified as redundant, calculate the variable assignment (if there exists one) that will generate a redundant instantiation of the two sets of rules.

The following example illustrates the algorithm: Consider the following two sets of rules, RR1 and RR2, that are reported as possibly redundant during the first phase of the algorithm;

RR1 consists of following rules:

R1: $\forall x \forall y [A(x, y) \wedge B(x, y) \rightarrow D(x, y)]$

R2: $\forall x \forall y [C(y, z) \wedge D(x, y) \rightarrow F(x, z)]$

RR2 consists of following rules:

R3: $\forall x \forall y \forall z [B(x, y) \wedge C(y, z) \rightarrow E(x, z)]$

R4: $\forall x \forall y \forall z [A(x, y) \wedge E(x, z) \rightarrow F(x, z)]$

The CPN representing these rules is shown in Figure 7.

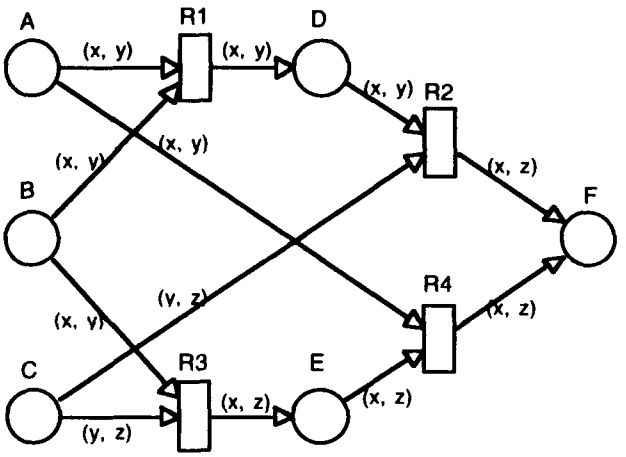

Figure 7 CPN Representation of Rules

The two steps of constructing the generalized deduction trees from the CPN in Figure 7 are illustrated in Figures 8 and 9. Figure 8 presents the generalized deduction tree constructed for the rules in set RR1, while Figure 9 shows the deduction tree for the rules in RR2.

The most general expression representing the rules in RR1 is obtained by collecting the leaves of the generalized deduction tree (Figure 8) as the premise and the root as the consequent. The rule obtained as a result is giver as:

$$A(x, y) \wedge B(x, y) \wedge C(y, z) \rightarrow F(x, z)$$

Similarly, the most general rule representing the set RR2 is given as:

$$A(x, y) \wedge B(x, y1) \wedge C(y1, z) \rightarrow F(x, z)$$

The terms of the consequent in the two consequents above are identical, however, a comparison of terms in the premises of the two rules reveals the fact that the expression representing RR2 is more general than the one for RR2, since the instances covered by the rules in RR1 represent a subset (for y = y1) of all the instances covered by the rules in RR2. Therefore, for the set of instances covered by rules in both the sets, the two sets represent a redundant case.

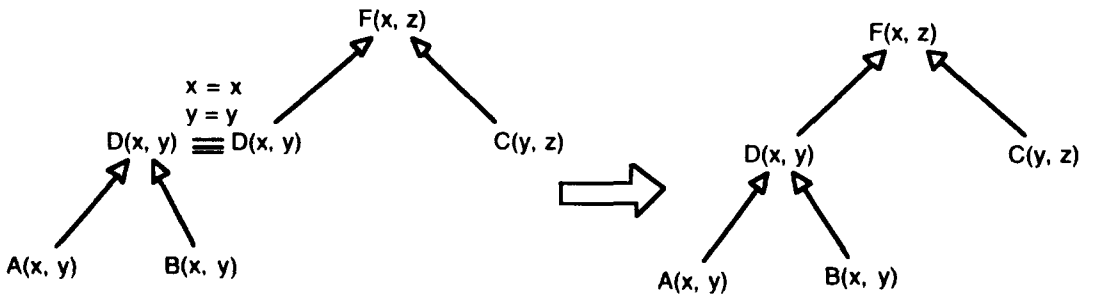

Figure 8 Generalized Deduction Tree For Rules In RR1

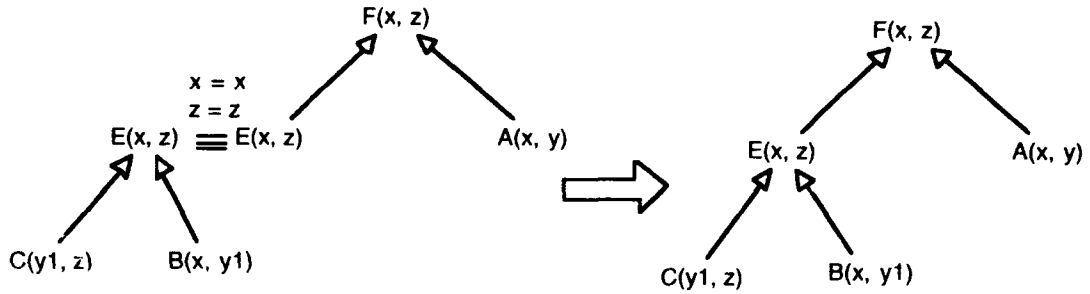

Figure 9 Generalized Deduction Tree For Rules In RR2

Algorithm for Subsumed Rules

(Two sets of rules SR1 and SR2 are found with structures representing subsumed cases — rules in SR1 are subsumed by the rules in SR2.)

- Construct a generalized deduction tree for rules in SR1. Construct another generalized tree for the rules in SR2.
- From the two generalized deduction trees, extract the most general form of the rules in the two sets by collecting the leaves of the tree as premises and the root as the consequent.
- Unify the variables in the terms in the two general expressions obtained. If unification results in a situation that the terms in expression representing SR2 are identical to the terms in SR1 corresponding to predicates common in both expressions, declare the rules as subsumed.
- In case the sets are not identified as subsumed, calculate the variable assignment (if there exists one) that will generate a subsumed instantiation of the two sets of rules.

The following example illustrates the algorithm: Consider two sets of rules, SR1 and SR2, where SR1 is reported to be possibly subsumed by the rules in SR2 during the first phase of the algorithm;

SR1 consists of following rules:

R1: $\forall x\, \forall y\, [A(x, y) \wedge B(x, y) \rightarrow D(x, y)]$

R2: $\forall x\, \forall y\, [C(y, x) \wedge D(x, y) \rightarrow F(x, y)]$

SR2 consists of following rule:

R3: $\forall u\, \forall v\, [A(u, v) \wedge B(u, v) \rightarrow F(u, v)]$

The CPN representing these rules is shown in Figure 10.

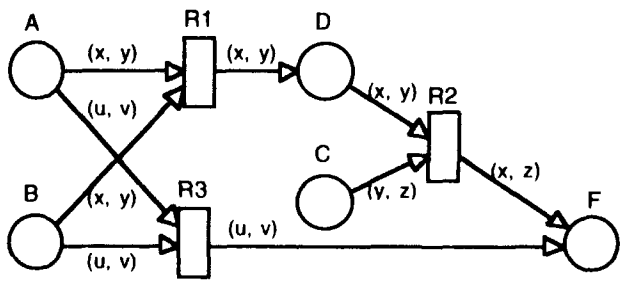

Figure 10 CPN Representation of Rules

Since there is only one rule in SR2, there is no need to construct the generalized deduction tree for SR2. The only rule in SR2, R3, is the most general expression representing the set. The generalized deduction tree for the rules in SR1 is shown in Figure 11. The generalized expression corresponding the set is given as:

$$A(x, y) \wedge B(x, y) \wedge C(y, x) \rightarrow F(x, y)$$

The unification of terms applied to the two expressions yields {x/u, y/v}.

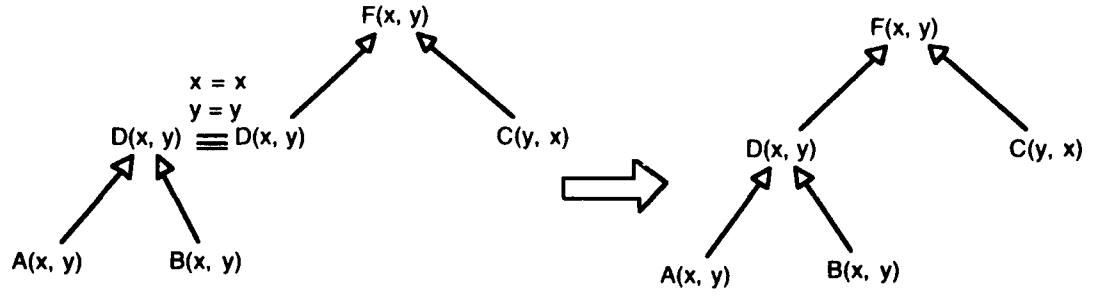

Figure 11 Generalized Deduction Tree For Rules In SR1

The terms in the rule representing SR2 when modified to reflect this unification make the rule look like

$$A(x, y) \wedge B(x, y) \rightarrow F(x, y)$$

which in fact subsumes the rules in SR1.

Algorithm for Conflicting Rules (Case II)

(Two sets of rules CR1 and CR2 are found with structures representing inconsistency — rules in CR1 are in conflict with the rules in CR2.)

- Construct a generalized deduction tree for rules in CR1. Construct another generalized tree for the rules in CR2.
- From the two generalized deduction trees, extract the most general form of the rules in the two sets by collecting the leaves of the tree as premises and the root as the consequent.
- Unify the variables in the terms in the two general expressions obtained. If unification results in two rules with identical premises and identical terms for the conflicting predicates in the consequents, declare the rules as conflicting (case II).
- In case the sets are not identified as conflicting, calculate the variable assignment (if there exists one) that will generate a conflicting instantiation of the two sets of rules.

As an example consider the following two sets of rules, CR1 and CR2, that are reported in conflict during the first phase of the algorithm;

CR1 consists of following rules:

R1: $\forall x \forall y [p1(x, y) \wedge p2(x) \wedge p3(y, x) \wedge p4(y) \rightarrow A(x, y)]$

CR2 consists of following rules:

R2: $\forall u \forall v [p1(u, v) \wedge p2(u) \rightarrow p5(u, v)]$

R3: $\forall x \forall y [p3(x, y) \wedge p4(x) \rightarrow p6(x, y)]$

R4: $\forall x \forall y [p5(x, y) \wedge p6(y, x) \rightarrow B(y, x)]$

where A and B are defined as mutually exclusive concepts; $\mu = \{A, B\}$.

The CPN representing these rules is shown in Figure 12.

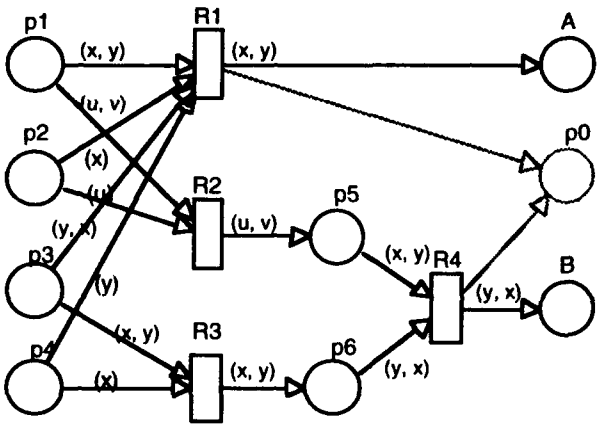

Figure 12 CPN Representation of Rules

Since there is only one rule in CR1, there is no need to construct a generalized deduction tree of CR1. The generalized deduction tree for the rules in CR2 is shown in Figure 13.

The most general rule representing the set CR2 is given as:

$$p1(x, y) \wedge p2(x) \wedge p3(y, x) \wedge p4(y) \rightarrow B(y, x)$$

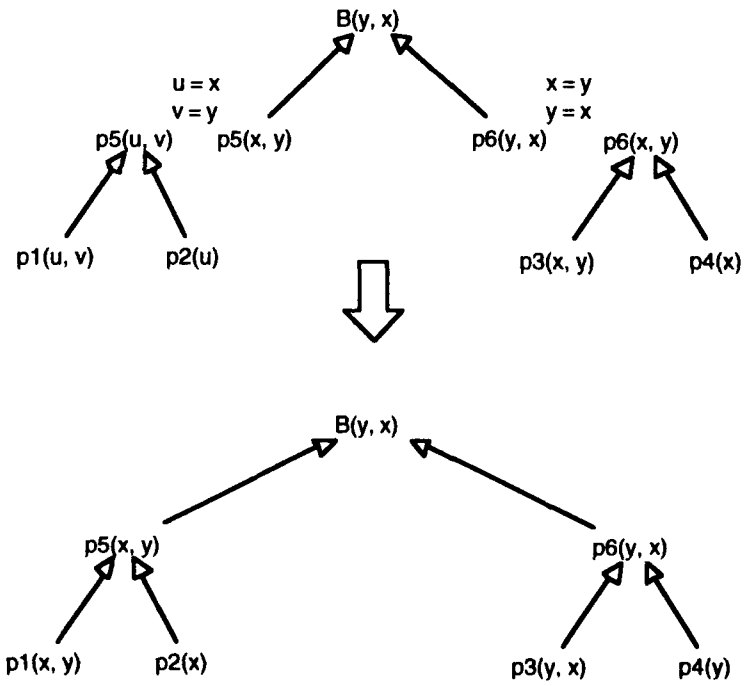

Figure 13 Generalized Deduction Tree For Rules In CR2

A comparison of the rule in CR1 with the general expression representing CR2 reveals that although the two premises are identical, but the consequents of the two rules are not in conflict provided the predicates A and B are not symmetric. In case the two concepts are symmetric, i.e., A(x, y) = B(y, x), the sets CR1 and CR2 are in conflict.

4.2 VARIABLE STRUCTURES: HEURISTIC RULES IN THE LATTICE ALGORITHM CONSTRAINTS

4.2.1 Including User Heuristics rules in the Lattice Algorithm

The lattice algorithm allows the automatic generation of candidate architectures satisfying a set of connection constraints and of structural constraints. The candidate architectures (which are numerous) are not listed singly but gathered in lattices defined by their largest and their smallest element: the maximally and minimally connected architectures (MAXOs and MINOs). Each candidate architecture belongs to one or more of the lattices and can be defined by addition of simple paths to the MINO or the subtraction of simple paths from the MAXO. The initial implementation of the work of Remy (1986) was limited to 5 Decision Making Units or DMUs (the size that could be handled by a 1 MB Macintosh Plus). More recent work by Zaidi (1992) has gone beyond this limitation by introducing a layered representation of the architecture, thus allowing for an arbitrary number of DMUs.

The input of the lattice algorithm is a set of connection constraints represented by matrices: e (input to the organization), s (output of the organization), F (SA to IF stages), G (RS to SA stages), H (RS to IF stages), and C (RS to CI stages). The architecture designer enters in each cell of the different matrices either "1" if he wants the corresponding connection to exist, "0" if he does not, or "2" if the corresponding link is optional. If n is the number of Decision Making Units (DMU) considered by the user, the initial number of elements in the set of possible structures is 2^{4n^2-2n}. For 5 DMUs, there are 2^{90} possible structures. Each time, the user specifies "1" or "0" in one cell of the matrices, the number of elements of the set of possible structures is divided by 2. After executing the Lattice Algorithm, the set of feasible structures has much fewer elements than the original set. The workload of the architecture's designer has been reduced because instead of looking at the entire set, he can concentrate only on the boundaries defined by the MINOs and MAXOs. When he has identified a pair of MINO and MAXO that contains interactions that seem appropriate for the specific application, he can further investigate the lattice connecting those two structures. The problem is that the number of feasible structures is still extensive and not easily manageable.

The set of feasible structures can be further reduced by taking into account in the Lattice Algorithm, heuristics, rules and requirements that are currently used later in the design process. Currently, the architecture designer has to specify a set of interactions, run the Lattice Algorithm to get the boundaries of the set of feasible structures defined by the MINOs and MAXOs, and use some kind of heuristics and rules to discard further feasible architectures to obtain a manageable set of architectures. On this final set, the designer can carry out some performance analysis to assess whether the obtained architectures satisfy a set of predetermined requirements. The focus of this task is to translate some of these rules, heuristics and requirements into additional constraints that become part of the input data in the Lattice Algorithm. Then these constraints will be satisfied in the automated architecture generation process.

After a brief description of the Lattice Algorithm, three types of additional constraints that could be included in the Lattice Algorithm are presented. The first one deals with the consideration of requirements on the degrees of redundancy and complexity desired in the architecture. The second one takes into account response time requirements. The last one addresses some user-defined rules that limit the types of interaction.

Brief Overview of the Lattice Algorithm

From the interaction constraints entered by the user in the different matrices, the algorithm generates two Petri nets: the Kernel Net corresponding to the structure deduced from the connection matrices where all the optional links are considered inactive and the Universal Net corresponding to the structures where all optional links are considered active.

In a second stage, the algorithm computes the S-invariants of the Universal Net and stores those that contain the source place and the sink place (the simple paths).

In a third stage, MINOs are generated by adding simple paths stored in the second stage to the Kernel Net until all the structural constraints are satisfied. Remy has defined four basic structural constraints:

R1 The structure should be connected

R2 The structure is acyclical (no loop)

R3 There exists at most one link from the RS stage of DM_i to the SA, IF, CI stage of DM_j: $G_{ij} + H_{ij} + C_{ij} \leq 1$

R4 Information Fusion takes place only at the IF and CI stages. The SA stage has at most one input with preference to external input: $e_j + G_{ij} \leq 1$

In a fourth stage, MAXOs are generated by keeping removing simple paths from the Universal Net until none of the constraints are violated.

Finally, the algorithm computes the invariants of the different MINOs and MAXOs to check the absence of loops. If there is a loop, the MINO or the MAXO is discarded.

Redundancy and Complexity

Redundancy and Complexity are two aspects of importance in the design of architecture. Redundancy is related to the dissemination of information in the organization for back-up purposes. While redundancy is desirable for survivability and reliability, it is limited by the communications network capacity. Complexity addresses the problem of fusion of information from different sources. The more sources, the more complex the fusion process and the more resource-consuming the process is. The architecture designer has to deal with these two parameters and perform some tradeoffs. The more redundancy is introduced, the more complex fusion algorithms will be or the more resources will be needed.

Remy (1986) gives an interpretation of MINOs and MAXOs. MINOs represent organization with a minimal number of interactions between DMUs. They represent not very survivable architectures because there is no redundancy to disseminate received or generated information throughout the system. However, they represent timely organizations. On the other hand, MAXOs represent architectures with a lot of redundancy, but that are less timely. An interesting feature for the Lattice Algorithm would be to take into account redundancy and complexity requirements in the generation process of feasible structures. Stamos Andreadakis

(1988) described an alternative methodology to generate architectures. After defining basic Information Flow Paths, he introduced the concepts of degree of Complexity at Fusion nodes and degree of Redundancy at Processing nodes. Redundancy is related to the number of Information Flow Paths that receive data generated from a Processing node while Complexity is related to the number of Information Flow Paths that send data to a Fusion Node. The definition of these degrees of redundancy and complexity is the basis for the generation of a set Data Flow Structures from which architectures can be derived by allocating functions to decision makers.

The concept of complexity and redundancy can be applied to the five stage model of the Decision Making Unit (Levis, 1993). The Situation Assessment and Response Selection stages can be considered as Processing nodes while Information Fusion and Command Interpretation stages are Fusion nodes. The task Processing stage has no external interactions so that it does not appear in the Lattice Algorithm; consequently, it will be ignored in what follows without any loss of generality. We can define SAR_i (resp. RSR_i) as the degree of redundancy of the SA (resp. RS) stage of DMU_i. SAR_i (resp. RSR_i) is equal to the number of output places of the transition representing the SA (resp. RS) stage of DMU_i. We can also define IFC_j (resp. CIC_j) as the degree of complexity of the IF (resp. CI) stage of DMU_j. IFC_j (resp. CIC_j) is equal to the number of input places of the transition representing the IF (resp. CI) stage of DMU_j. If n is the number of DMUs, using the connection matrices of the Lattice Algorithm , these degrees can expressed as:

$$SAR_i = \begin{cases} 1 + \sum_{j=1}^{n} F_{ij} & \text{if stage IFi exists} \\ \sum_{j=1}^{n} F_{ij} & \text{if stage IFi does not exist} \end{cases}$$

$$RSR_i = s_i + \sum_{j=1}^{n} G_{ij} + H_{ij} + C_{ij}$$

$$IFC_j = \begin{cases} 1 + \sum_{i=1}^{n} F_{ij} + H_{ij} & \text{if stage SAj exists} \\ \sum_{i=1}^{n} F_{ij} + H_{ij} & \text{if stage SAj does not exist} \end{cases}$$

$$CIC_i = \begin{cases} 1 + \sum_{i=1}^{n} C_{ij} & \text{if stage IFj exists} \\ \sum_{i=1}^{n} C_{ij} & \text{if stage IFj does not exist} \end{cases}$$

The Lattice Algorithm uses a specific labeling for each place and for each transition as shown on Figure 14. The computation of the S-invariants results in the generation of 1 x m vectors. Each entry corresponds to a place of the Universal Net and takes the value 1, if the place is included in the S-invariant, 0 if it is not. This vector representation is also used for structures obtained by adding (joining) simple paths. The join operation is noted by $\cup$. and we have:

$[X_1 \ldots X_i \ldots X_n] \cup [Y_1 \ldots Y_i \ldots Y_n] = [(X_1 \text{ or } Y_1), \ldots (X_i \text{ or } Y_i) \ldots (X_n \text{ or } Y_n)]$

where $X_i, Y_i \in \{0,1\}$ and X_i or $Y_i = 0$ if $X_i = 0$ and $Y_i = 0$
X_i or $Y_i = 1$ otherwise.

Thus, a structure is also represented by a vector, each entry corresponding to a place of the Universal Net and taking value 1 if the place is included in the structure, 0 if it is not. Therefore, it is possible to define the degrees of redundancy and complexity of a structure in term of the values taken by the entries of its vector representation. If we denote by $P_{xyzt}(s)$ the function that gives the value of the entry corresponding to place P_{xyzt} in the vector representation of the structure s, we have:

$$SAR_i(s) = P_{2i}(s) + \sum_{\substack{j=1 \\ j\neq i}}^{n} P_{2ij}(s)$$

$$RSR_i(s) = P_{5i}(s) + \sum_{\substack{j=1 \\ j\neq i}}^{n} P_{5ij1}(s) + P_{5ij2}(s) + P_{5ij3}(s)$$

$$IFC_j(s) = P_{2j}(s) + \sum_{\substack{i=1 \\ i\neq j}}^{n} P_{2ij}(s) + P_{5ij2}(s)$$

$$CIC_j(s) = P_{3j}(s) + \sum_{\substack{i=1 \\ i\neq j}}^{n} P_{5ij3}(s)$$

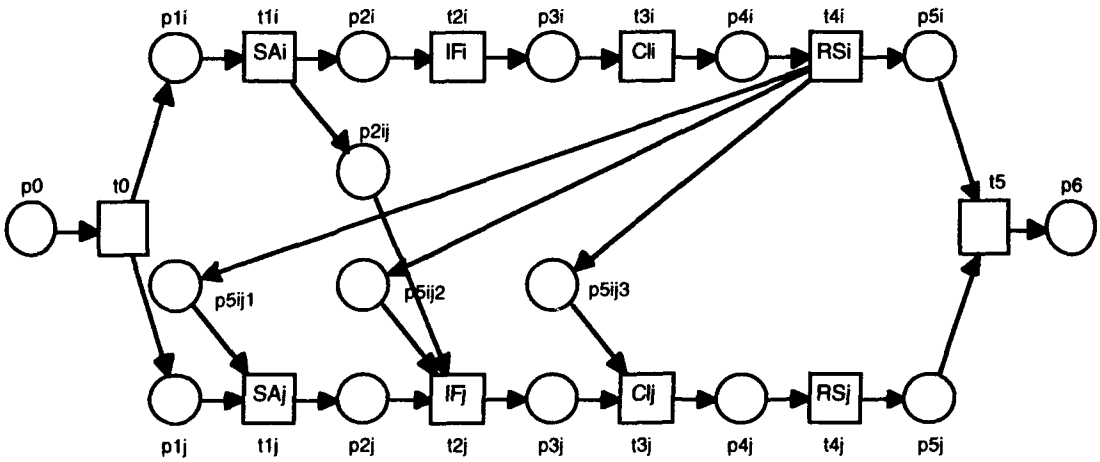

Figure 14 Labeling of Places and Transitions in the Lattice Algorithm

The architecture designer can specify ranges for the different degrees of redundancy and complexity for the architecture he envisions. These ranges will introduce additional constraints that the Lattice algorithm will check in the same way as it checks constraints R3 and R4.

The different degrees can not be fixed arbitrarily because they are functions of each others: The definition minimal degree of redundancy requires the definition of minimal degree of complexity to make sure that each interaction place created to meet the redundancy requirements will not violate the complexity requirements. The first constraint is that if n is the number of decision making units, the different degrees have to satisfy:

$$0 \leq SAR_i \leq n \quad 0 \leq RSR_i \leq n \quad 0 \leq IFC_j \leq n \quad 0 \leq CIC_j \leq n$$

On the other hand, we have:

$$\sum_{i=1}^{n} SAR_i + \sum_{i=1}^{n} RSR_i = \sum_{i=1}^{n} P_{2i} + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} P_{2ij} + \sum_{i=1}^{n} P_{5i} + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} P_{5ij1} + P_{5ij2} + P_{5ij3}$$

$$\sum_{j=1}^{n} IFC_j + \sum_{j=1}^{n} CIC_j = \sum_{j=1}^{n} P_{2j} + \sum_{j=1}^{n} \sum_{\substack{i=1 \\ i \neq j}}^{n} (P_{2ij} + P_{5ij2}) + \sum_{j=1}^{n} P_{3j} + \sum_{j=1}^{n} \sum_{\substack{i=1 \\ i \neq j}}^{n} P_{5ij3}$$

and therefore:

$$\sum_{i=1}^{n} SAR_i + \sum_{i=1}^{n} RSR_i = \sum_{i=1}^{n} IFC_i + \sum_{i=1}^{n} CIC_i - \sum_{i=1}^{n} P_{3i} + \sum_{i=1}^{n} P_{5i} + \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} P_{5ij1}$$

This relation between the different degrees is much more complex than the one derived by Andreadakis. It can not be used before running the Lattice Algorithm because, once the ranges of the degrees have been specified, it is impossible to know before hand, if places P_{3i}, P_{5i} and P_{5ij1} exist or not.

It is easy to show that these new constraints are convex. The addition of a simple path to a structure will only increase some of those different degrees by 1. If s is a structure and p is a simple path, we will have:

$\forall i \in \{1, ..., n\}$, $SAR_i(s) \leq SAR_i(s \cup p) \leq SAR_i(s) + 1$

$\forall i \in \{1, ..., n\}$, $RSR_i(s) \leq RSR_i(s \cup p) \leq RSR_i(s) + 1$

$\forall j \in \{1, ..., n\}$, $IFC_j(s) \leq IFC_j(s \cup p) \leq IFC_j(s) + 1$

$\forall j \in \{1, ..., n\}$, $CIC_j(s) \leq CIC_j(s \cup p) \leq CIC_j(s) + 1$

Therefore, if we denote by $\subset$ the operation that satisfies:

$[X_1 \ ... \ X_i \ ... \ X_m] \subset [Y_1 \ ... \ Y_i \ ... \ Y_m]$ if $\forall i \in \{1, ..., m\}$, $Xi \leq Yi$

it is easy to deduce that if $s1 \subset s2$ then:

$\forall i \in \{1, ..., n\}$, $SAR_i(s1) \leq SAR_i(s\ 2)$

$\forall i \in \{1, ..., n\}$, $RSR_i(s1) \leq RSR_i(s2)$

$\forall j \in \{1, ..., n\}$, $IFC_j(s1) \leq IFC_j(s2)$

$\forall j \in \{1, ..., n\}$, $CIC_j(s1) \leq CIC_j(s2)$

and we have for all s such that $s1 \subset s \subset s2$:

$\forall i \in \{1, ..., n\}$, $SAR_i(s1) \leq SAR_i(s\ ) \leq SAR_i(s2)$

$\forall i \in \{1, ..., n\}$, $RSR_i(s1) \leq RSR_i(s) \leq RSR_i(s2)$

$\forall j \in \{1, ..., n\}$, $IFC_j(s1) \leq IFC_j(s) \leq IFC_j(s2)$

$\forall j \in \{1, ..., n\}$, $CIC_j(s1) \leq CIC_j(s) \leq CIC_j(s2)$

From the convexity properties we have just addressed, we can deduce that there exist feasible architectures that satisfy all the constraints if and only if there exists a MINO for which the different degrees of complexity and redundancy are smaller than the upper boundaries defined by the architecture designer and there exists a MAXO for which these degrees are larger than the lower boundaries.

The output of the updated Lattice Algorithm will be a new set of MINOs and MAXOs that will define boundaries of lattices included in the ones generated by the Lattice Algorithm without these new constraints (if these constraints are binding).

Let us illustrate these concepts in an example from Remy (1986): the Warfare Commander Problem. This organization has 5 DMUs. DMU_1 and DMU_2 act as the sensors of the organization (Sonar Operator (SO) and Radar Operator (RO) for example). They both receive information from the external environment (threat detection). They may or may not share this information. However, DMU_1 has to send this information to DMU_3 which acts as the Executive Coordinator (EXCO) . Finally, DMU_4, the Anti-Air Warfare Commander (AAWC) and DMU_5, the Anti-Submarine Warfare Commander (ASWC) produce the organization's response (firing of missiles for AAWC or torpedoes and depth charges for ASWC). They receive orders from the coordinator DMU_3 and may receive information from DMU_1 and DMU_2. They may also share their results. The connection matrices corresponding to this description of the organization is displayed in Figure 15.

e

ext -> SA

1	1	0	0	0

s

RS -> ext

0	0	0	1	1

F

SA -> IF

X	2	0	0	0
2	X	0	0	0
0	0	X	0	0
0	0	0	X	0
0	0	0	0	X

G

RS -> SA

X	0	0	0	0
0	X	0	0	0
0	0	X	0	0
0	0	0	X	0
0	0	0	0	X

H

RS -> IF

X	0	1	2	2
0	X	2	2	2
0	0	X	0	0
0	0	2	X	2
0	0	2	2	X

C

RS -> CI

X	0	0	0	0
0	X	0	0	0
0	0	X	1	2
0	0	0	X	0
0	0	0	0	X

Figure 15 Connection matrices for the Example

The Lattice Algorithm computes 40 invariants and produces 10 MINOs and 3 MAXOs that are listed in Table 1. Table 1 lists also the different degrees of redundancy and complexity computed with the formula defined above.

Table 1 List of MINOs and MAXOs with their degrees of redundancy and complexity

		MINOS										MAXOS		
	PLACES	1	2	3	4	5	6	7	8	9	10	1	2	3
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
2	11	1	1	1	1	1	1	1	1	1	1	1	1	1
3	12	1	1	1	1	1	1	1	1	1	1	1	1	1
4	21	1	1	1	1	1	1	1	1	1	1	1	1	1
5	212	0	0	0	0	0	0	0	0	0	0	1	1	1
6	221	0	0	0	1	1	1	0	0	0	0	1	1	1
7	22	1	1	1	0	0	0	1	1	1	1	1	1	1
8	31	1	1	1	1	1	1	1	1	1	1	1	1	1
9	32	1	1	1	0	0	0	1	1	1	1	1	1	1
10	33	1	1	1	1	1	1	1	1	1	1	1	1	1
11	34	1	1	1	0	0	0	0	0	0	0	1	1	1
12	35	1	0	1	1	0	1	1	0	1	1	1	1	1
13	41	1	1	1	1	1	1	1	1	1	1	1	1	1
14	42	1	1	1	0	0	0	1	1	1	1	1	1	1
15	43	1	1	1	1	1	1	1	1	1	1	1	1	1
16	44	1	1	1	1	1	1	1	1	1	1	1	1	1
17	45	1	1	1	1	1	1	1	1	1	1	1	1	1
18	5132	1	1	1	1	1	1	1	1	1	1	1	1	1
19	5142	0	0	0	0	0	0	0	0	0	0	1	1	1
20	5152	1	0	0	1	0	0	1	0	0	0	1	1	1
21	5232	0	0	0	0	0	0	1	1	1	0	1	1	1
22	5242	1	1	1	0	0	0	0	0	0	0	1	1	1
23	5252	0	0	0	0	0	0	0	0	0	1	1	1	1
24	5343	1	1	1	1	1	1	1	1	1	1	1	1	1
25	5353	0	1	0	0	1	0	0	1	0	0	0	1	1
26	54	1	1	1	1	1	1	1	1	1	1	1	1	1
27	5432	0	0	0	0	0	0	0	0	0	0	0	0	0
28	5452	0	0	1	0	0	1	0	0	1	0	0	0	1
29	55	1	1	1	1	1	1	1	1	1	1	1	1	1
30	5532	0	0	0	0	0	0	0	0	0	0	1	0	0
31	5542	0	0	0	0	0	0	0	0	0	0	1	1	0
32	6	1	1	1	1	1	1	1	1	1	1	1	1	1
	SAR1	1	1	1	1	1	1	1	1	1	1	2	2	2
	SAR2	1	1	1	1	1	1	1	1	1	1	2	2	2
	SAR3	0	0	0	0	0	0	0	0	0	0	0	0	0
	SAR4	0	0	0	0	0	0	0	0	0	0	0	0	0
	SAR5	0	0	0	0	0	0	0	0	0	0	0	0	0
	RSR1	2	1	1	2	1	1	2	1	1	1	3	3	3
	RSR2	1	1	1	0	0	0	1	1	1	1	3	3	3
	RSR3	1	2	1	1	2	1	1	2	1	1	1	2	2
	RSR4	1	1	2	1	1	2	1	1	2	1	1	1	2
	RSR5	1	1	1	1	1	1	1	1	1	1	3	2	1
	IFC1	1	1	1	2	2	2	1	1	1	1	2	2	2
	IFC2	1	1	1	0	0	0	1	1	1	1	2	2	2
	IFC3	1	1	1	1	1	1	2	2	2	1	3	2	2
	IFC4	1	1	1	0	0	0	0	0	0	0	3	3	2
	IFC5	1	0	1	1	0	1	1	0	1	1	2	2	3
	CIC1	1	1	1	1	1	1	1	1	1	1	1	1	1
	CIC2	1	1	1	0	0	0	1	1	1	1	1	1	1
	CIC3	1	1	1	1	1	1	1	1	1	1	1	1	1
	CIC4	2	2	2	1	1	1	1	1	1	1	2	2	2
	CIC5	1	1	1	1	1	1	1	1	1	1	1	2	2

Let us assume that in order to meet communications, processing and survivability requirements, the architecture designer specifies that:

$SAR_1 = 2$	$SAR_2 = 2$			
$RSR_1 \leq 2$	$RSR_2 \leq 2$	$RSR_3 = 2$	$RSR_4 = 1$	$RSR_5 = 1$
$IFC_1 = 2$	$IFC_2 = 2$	$IFC_3 = 2$	$IFC_4 \leq 1$	$IFC_5 \leq 1$
$CIC_1 = 1$	$CIC_2 = 1$	$CIC_3 = 1$	$CIC_4 = 2$	$CIC_5 = 2$

From Table 1, we can conclude that none of the MINOs satisfies the architecture designer requirements because SAR_1 and SAR_2 for those MINOs are all less than or equal to 1 and none of the MAXOs satisfies those requirements because IFC_4 and IFC_5 for the MAXOs are larger than or equal to 2. However, a set of feasible structures exists because there exists a MINO (for example MINO 8) for which the degrees of redundancy and complexity are smaller than the maximal requirements. and there exists a MAXO (for example M2) for which the degrees of redundancy and complexity are larger than the minimal requirements. To get the new set of MINOs and MAXOs that satisfy these requirements, more simple paths need to be added to the current MINOs and other simple paths need to be subtracted from the MAXOs. Let us consider the MINO m8 and MAXO M2 displayed in Figure 16 and 17. We have $m8 \subset M2$.

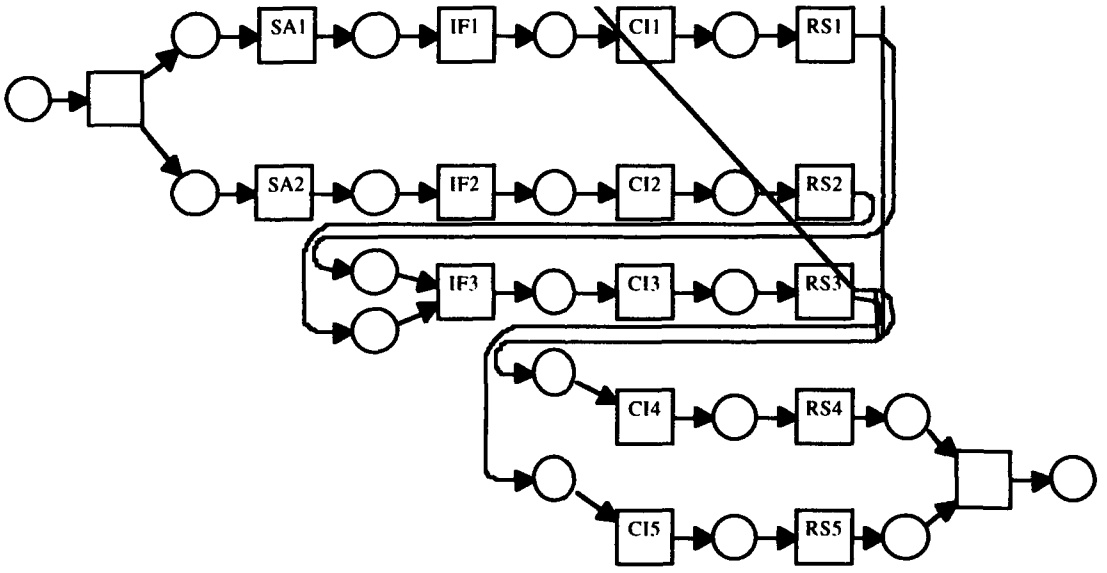

Figure 16 MINO m8

MINO m8 does not satisfy the constraints $SAR_1 = 2$ $SAR_2 = 2$ $IFC_1 = 2$ and $IFC_2 = 2$. Adding a simple path containing the interaction place between SA1 and IF2 (p122) and places that are already in the structure will increase SAR_1 and IFC_2 by 1 and leave the other degrees unchanged. In the same way, adding a simple path containing the interaction place between SA2 and IF1 (p221) and places that are already in the structure will increase SAR_2 and IFC_1 by 1 and leave the other degrees unchanged. Adding these two appropriate simple paths will result in a MINO for the complete set of constraints. This new MINO is displayed in Figure 18.

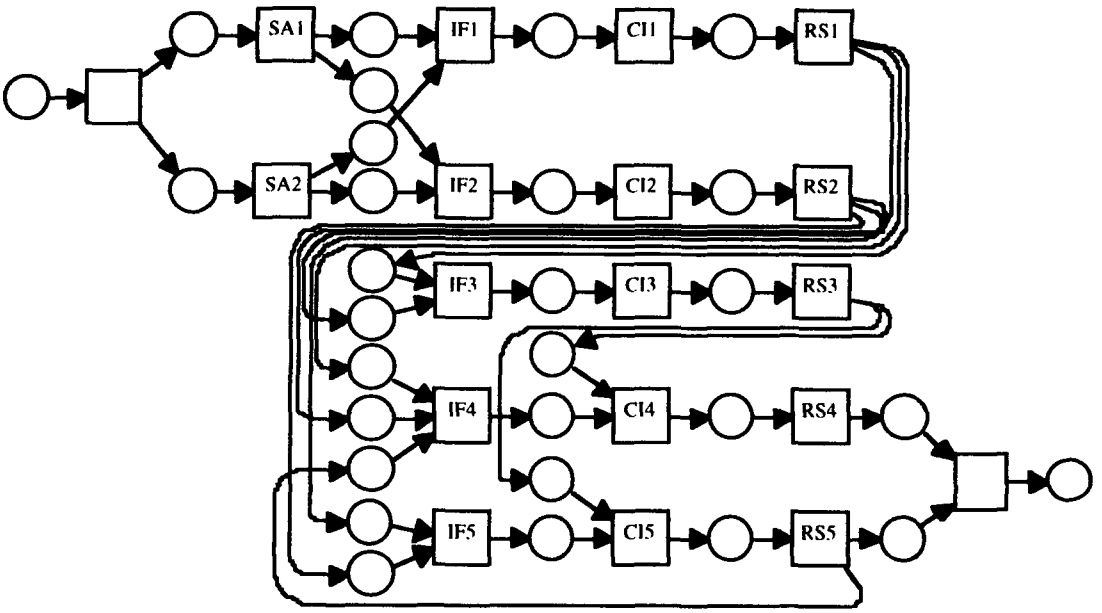

Figure 17 MAXO M2

MAXO M2 violates the constraints $RSR_1 \leq 2$, $RSR_2 \leq 2$, $RSR_5 = 1$, $IFC_4 \leq 1$, and $IFC_5 \leq 1$ Here again, by removing appropriate simple paths that contain undesirable interaction places, a MAXOs satisfying the complete set of constraints can be reached. Figure 19 displays one of the new MAXO that can be reached by removing simple paths.

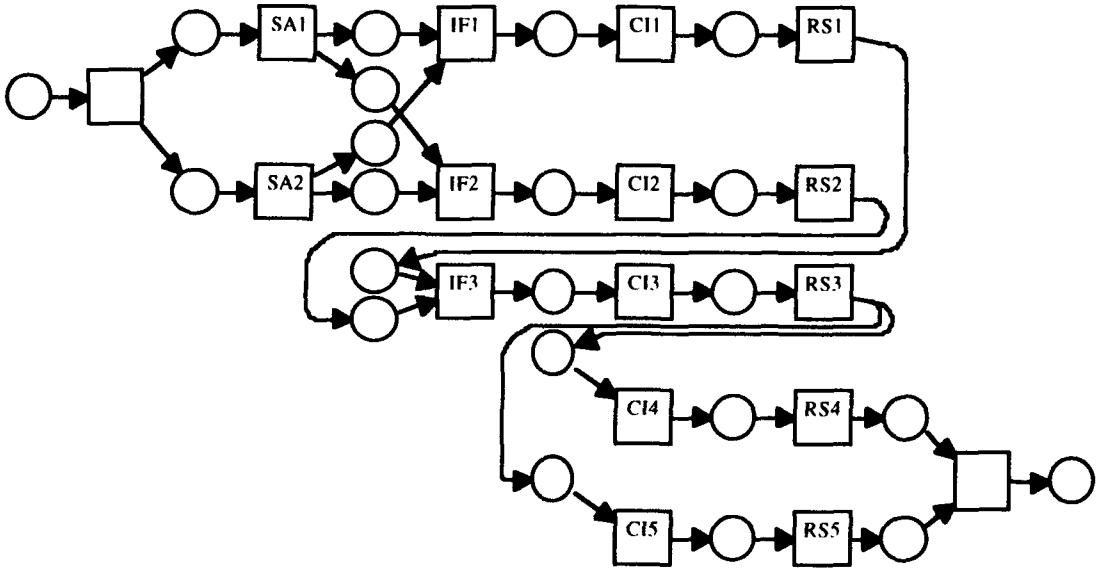

Figure 18 New MINO Reached by Adding Simple Paths to m8

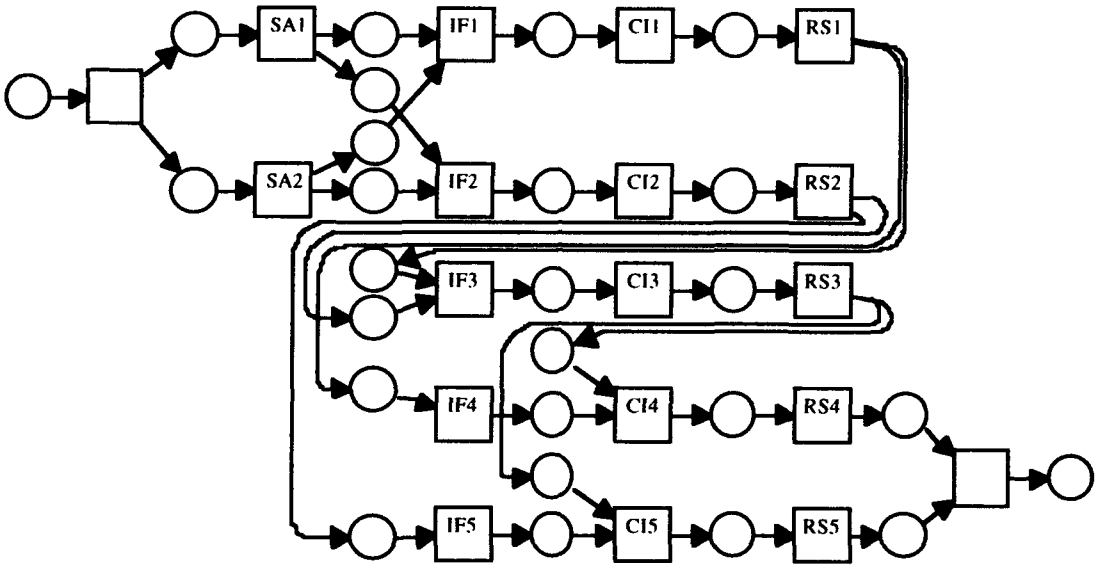

Figure 19 New MAXO Reached by Removing Simple Paths from M2

The process we have just described is the process used by the Lattice Algorithm to generate feasible architecture: it keeps adding simple paths to the Kernel net until all the constraints are satisfied to generate the set of MINOs and it keeps subtracting Simple Path from the Universal Net until none of the constraints are violated. These new constraints on complexity and redundancy will be easy to include in the software application of the algorithm since it will not change the global process of MINO and MAXO generation.

Response Time
Response time is a critical measure of performance that is always considered by the architecture designer in selecting the final set of candidate structures. Timed Petri Nets are a tool that allows to assess if the response time of an architecture will be less than a given time requirement. This new constraint can be handled easily by the Lattice Algorithm.

Once the source and sink places are merged together, the Petri Nets handled by the Lattice Algorithm are marked graphs. Hillion (1986) showed that the response time of the system is equal to the largest sum of the delays of the transitions of the S-component of the S-invariants of the Petri Net. Therefore, the first step is to associate a delay with each transition that corresponds to the amount of time required to perform the process the transition represents.

The second step is to compute the delay associated with each Simple Path computed by the Lattice Algorithm and discard those whose delay is larger than the required response time. The lattice algorithm will then be executed with this subset of Simple Paths and a smaller set of solutions will be generated. *All the solutions of this set will satisfy the requirements.*

Let us describe the procedure on the Warfare Commander example described in the previous section. As said earlier, the Lattice Algorithm generates 40 simple paths / invariants for the connection constraints specified in Figure 15. The minimal support of these 40 invariants are listed in Table 2.

Table 2 Minimal Support of the Warfare Commander Problem

sp	Minimal Support
1	p_0-t_0-p_{11}-t_{11}-p_{21}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5142}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6
2	p_0-t_0-p_{12}-t_{12}-p_{221}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5142}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6
3	p_0-t_0-p_{11}-t_{11}-p_{212}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5242}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6
4	p_0-t_0-p_{12}-t_{12}-p_{22}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5242}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6
5	p_0-t_0-p_{11}-t_{11}-p_{21}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5132}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5343}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6
6	p_0-t_0-p_{12}-t_{12}-p_{221}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5132}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5343}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6
7	p_0-t_0-p_{11}-t_{11}-p_{212}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5232}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5343}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6
8	p_0-t_0-p_{12}-t_{12}-p_{22}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5232}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5343}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6
9	p_0-t_0-p_{11}-t_{11}-p_{21}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5152}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
10	p_0-t_0-p_{12}-t_{12}-p_{221}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5152}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
11	p_0-t_0-p_{11}-t_{11}-p_{212}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5252}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
12	p_0-t_0-p_{12}-t_{12}-p_{22}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5252}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
13	p_0-t_0-p_{11}-t_{11}-p_{21}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5132}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5353}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
14	p_0-t_0-p_{12}-t_{12}-p_{221}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5132}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5353}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
15	p_0-t_0-p_{11}-t_{11}-p_{212}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5232}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5353}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
16	p_0-t_0-p_{12}-t_{12}-p_{22}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5232}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5353}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
17	p_0-t_0-p_{11}-t_{11}-p_{21}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5142}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{5452}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
18	p_0-t_0-p_{12}-t_{12}-p_{221}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5142}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{5452}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
19	p_0-t_0-p_{11}-t_{11}-p_{212}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5242}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{5452}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
20	p_0-t_0-p_{12}-t_{12}-p_{22}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5242}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{5452}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
21	p_0-t_0-p_{11}-t_{11}-p_{21}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5132}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5343}-t_{34}-p_{44}-t_{44}-p_{5452}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
22	p_0-t_0-p_{12}-t_{12}-p_{221}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5132}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5343}-t_{34}-p_{44}-t_{44}-p_{5452}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
23	p_0-t_0-p_{11}-t_{11}-p_{212}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5232}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5343}-t_{34}-p_{44}-t_{44}-p_{5452}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
24	p_0-t_0-p_{12}-t_{12}-p_{22}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5232}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5343}-t_{34}-p_{44}-t_{44}-p_{5452}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
25	p_0-t_0-p_{11}-t_{11}-p_{21}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5142}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{5432}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5353}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
26	p_0-t_0-p_{12}-t_{12}-p_{221}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5142}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{5432}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5353}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
27	p_0-t_0-p_{11}-t_{11}-p_{212}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5242}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{5432}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5353}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
28	p_0-t_0-p_{12}-t_{12}-p_{22}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5242}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{5432}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5353}-t_{35}-p_{45}-t_{45}-p_{55}-t_5-p_6
29	p_0-t_0-p_{11}-t_{11}-p_{21}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5152}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{5542}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6
30	p_0-t_0-p_{12}-t_{12}-p_{221}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5152}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{5542}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6
31	p_0-t_0-p_{11}-t_{11}-p_{212}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5252}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{5542}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6
32	p_0-t_0-p_{12}-t_{12}-p_{22}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5252}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{5542}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6
33	p_0-t_0-p_{11}-t_{11}-p_{21}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5132}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5353}-t_{35}-p_{45}-t_{45}-p_{5542}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6
34	p_0-t_0-p_{12}-t_{12}-p_{221}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5132}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5353}-t_{35}-p_{45}-t_{45}-p_{5542}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6
35	p_0-t_0-p_{11}-t_{11}-p_{212}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5232}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5353}-t_{35}-p_{45}-t_{45}-p_{5542}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6
36	p_0-t_0-p_{12}-t_{12}-p_{22}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5232}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5353}-t_{35}-p_{45}-t_{45}-p_{5542}-t_{24}-p_{34}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6
37	p_0-t_0-p_{11}-t_{11}-p_{21}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5152}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{5532}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5343}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6
38	p_0-t_0-p_{12}-t_{12}-p_{221}-t_{21}-p_{31}-t_{31}-p_{41}-t_{41}-p_{5152}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{5532}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5343}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6
39	p_0-t_0-p_{11}-t_{11}-p_{212}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5252}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{5532}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5343}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6
40	p_0-t_0-p_{12}-t_{12}-p_{22}-t_{22}-p_{32}-t_{32}-p_{42}-t_{42}-p_{5252}-t_{25}-p_{35}-t_{35}-p_{45}-t_{45}-p_{5532}-t_{23}-p_{33}-t_{33}-p_{43}-t_{43}-p_{5343}-t_{34}-p_{44}-t_{44}-p_{54}-t_5-p_6

If we specify that the response time of the organization has to be less than 80 seconds and that the delays associated with each processes are defined as follows:

$t_0 = 0$

$t_{SA1} = t_{11} = 10$	$t_{IF1} = t_{21} = 5$	$t_{CI1} = t_{31} = 0$	$t_{RS1} = t_{41} = 10$
$t_{SA2} = t_{12} = 10$	$t_{IF2} = t_{22} = 5$	$t_{CI1} = t_{32} = 0$	$t_{RS2} = t_{42} = 10$
	$t_{IF3} = t_{23} = 10$	$t_{CI3} = t_{33} = 0$	$t_{RS3} = t_{43} = 10$
	$t_{IF4} = t_{24} = 10$	$t_{CI4} = t_{34} = 5$	$t_{RS4} = t_{44} = 10$
	$t_{IF5} = t_{25} = 10$	$t_{CI5} = t_{35} = 5$	$t_{RS5} = t_{45} = 10$

$t_5 = 0$

The delays associated with each simple path are:

T1 = t0 + t11 + t21 + t31 + t41 + t24 + t34 + t44 + t5 = 50
T2 = t0 + t12 + t21 + t31 + t41 + t24 + t34 + t44 + t5 = 50
T3 = t0 + t11 + t22 + t32 + t42 + t24 + t34 + t44 + t5 = 50
T4 = t0 + t12 + t22 + t32 + t42 + t24 + t34 + t44 + t5 = 50
T5 = t0 + t11 + t21 + t31 + t41 + t23 + t33 + t43 + t34 + t44 + t5 = 60
T6 = t0 + t12 + t21 + t31 + t41 + t23 + t33 + t43 + t34 + t44 + t5 = 60
T7 = t0 + t11 + t22 + t32 + t42 + t23 + t33 + t43 + t34 + t44 + t5 = 60
T8 = t0 + t12 + t22 + t32 + t42 + t23 + t33 + t43 + t34 + t44 + t5 = 60
T9 = t0 + t11 + t21 + t31 + t41 + t25 + t35 + t45 + t5 = 50
T10 = t0 + t12 + t21 + t31 + t41 + t25 + t35 + t45 + t5 = 50
T11 = t0 + t11 + t22 + t32 + t42 + t25 + t35 + t45 + t5 = 50
T12 = t0 + t12 + t22 + t32 + t42 + t25 + t35 + t45 + t5 = 50
T13 = t0 + t11 + t21 + t31 + t41 + t23 + t33 + t43 + t35 + t45 + t5 = 60
T14 = t0 + t12 + t21 + t31 + t41 + t23 + t33 + t43 + t35 + t45 + t5 = 60
T15 = t0 + t11 + t22 + t32 + t42 + t23 + t33 + t43 + t35 + t45 + t5 = 60
T16 = t0 + t12 + t22 + t32 + t42 + t23 + t33 + t43 + t35 + t45 + t5 = 60
T17 = t0 + t11 + t21 + t31 + t41 + t24 + t34 + t44 + t25 + t35 + t45 + t5 = 75
T18 = t0 + t12 + t21 + t31 + t41 + t24 + t34 + t44 + t25 + t35 + t45 + t5 = 75
T19 = t0 + t11 + t22 + t32 + t42 + t24 + t34 + t44 + t25 + t35 + t45 + t5 = 75
T20 = t0 + t12 + t22 + t32 + t42 + t24 + t34 + t44 + t25 + t35 + t45 + t5 = 75
T21 = t0 + t11 + t21 + t31 + t41 + t23 + t33 + t43 + t34 + t44 + t25 + t35 + t45 + t5 = 85
T22 = t0 + t12 + t21 + t31 + t41 + t23 + t33 + t43 + t34 + t44 + t25 + t35 + t45 + t5 = 85
T23 = t0 + t11 + t22 + t32 + t42 + t23 + t33 + t43 + t34 + t44 + t25 + t35 + t45 + t5 = 85
T24 = t0 + t12 + t22 + t32 + t42 + t23 + t33 + t43 + t34 + t44 + t25 + t35 + t45 + t5 = 85
T25 = t0 + t11 + t21 + t31 + t41 + t24 + t34 + t44 + t23 + t33 + t43 + t35 + t45 + t5 = 85
T26 = t0 + t12 + t21 + t31 + t41 + t24 + t34 + t44 + t23 + t33 + t43 + t35 + t45 + t5 = 85
T27 = t0 + t11 + t22 + t32 + t42 + t24 + t34 + t44 + t23 + t33 + t43 + t35 + t45 + t5 = 85
T28 = t0 + t12 + t22 + t32 + t42 + t24 + t34 + t44 + t23 + t33 + t43 + t35 + t45 + t5 = 85
T29 = t0 + t11 + t21 + t31 + t41 + t25 + t35 + t45 + t24 + t34 + t44 + t5 = 75
T30 = t0 + t12 + t21 + t31 + t41 + t25 + t35 + t45 + t24 + t34 + t44 + t5 = 75
T31 = t0 + t11 + t22 + t32 + t42 + t25 + t35 + t45 + t24 + t34 + t44 + t5 = 75
T32 = t0 + t12 + t22 + t32 + t42 + t25 + t35 + t45 + t24 + t34 + t44 + t5 = 75
T33 = t0 + t11 + t21 + t31 + t41 + t23 + t33 + t43 + t35 + t45 + t24 + t34 + t44 + t5 = 85
T34 = t0 + t12 + t21 + t31 + t41 + t23 + t33 + t43 + t35 + t45 + t24 + t34 + t44 + t5 = 85
T35 = t0 + t11 + t22 + t32 + t42 + t23 + t33 + t43 + t35 + t45 + t24 + t34 + t44 + t5 = 85
T36 = t0 + t12 + t22 + t32 + t42 + t23 + t33 + t43 + t35 + t45 + t24 + t34 + t44 + t5 = 85
T37 = t0 + t11 + t21 + t31 + t41 + t25 + t35 + t45 + t23 + t33 + t43 + t34 + t44 + t5 = 85
T38 = t0 + t12 + t21 + t31 + t41 + t25 + t35 + t45 + t23 + t33 + t43 + t34 + t44 + t5 = 85
T39 = t0 + t11 + t22 + t32 + t42 + t25 + t35 + t45 + t23 + t33 + t43 + t34 + t44 + t5 = 85
T40 = t0 + t12 + t22 + t32 + t42 + t25 + t35 + t45 + t23 + t33 + t43 + t34 + t44 + t5 = 85

One can see that 14 simple paths: 21, 22, 23, 24, 25, 26, 27, 28, 33, 34, 35, 36, 37,38, 39, and 40 have an associated delay of 85 seconds, and therefore any generated structure containing at least one of these simple paths will have a response time of 85s which is larger than the 80 seconds required. These simple paths need therefore to be discarded and the Lattice Algorithm will construct the feasible structures by combining the 26 remaining simple paths.

The resulting set of solutions will be smaller. For example, MAXO M2 displayed in Figure 17 contains Simple Paths 33, 34, 35 and 36 that have been discarded. The violation of the response time requirements comes from the fact that DMU 4 has to wait for DMU5 to produce its response before starting its task. By subtracting these Simple Paths, a new MAXO M'2 satisfying the response time requirement is obtained and is displayed in Figure 20.

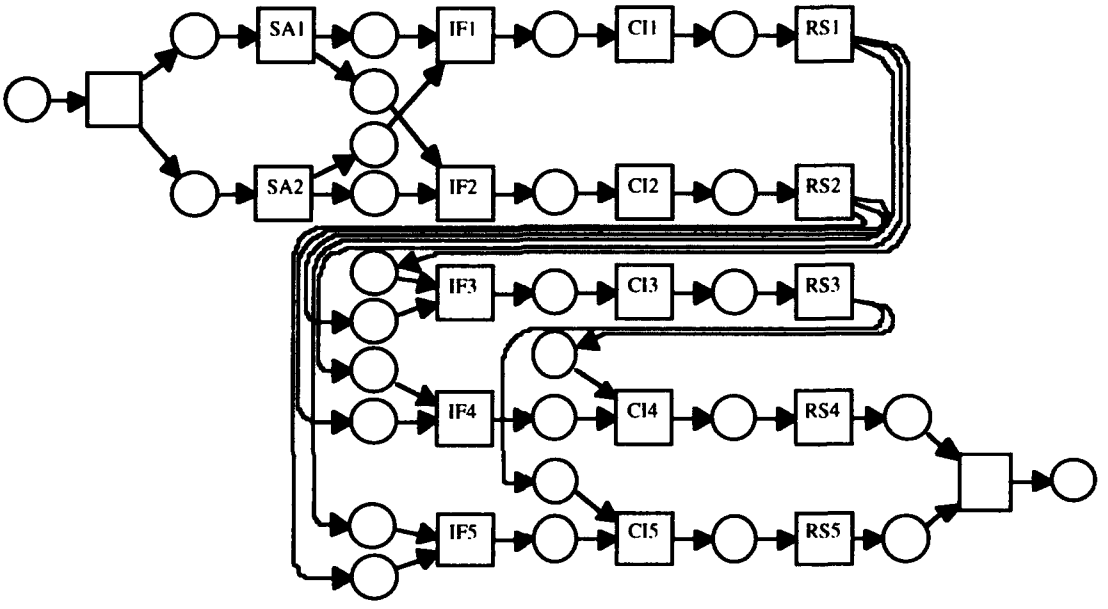

Figure 20 New MAXO M'2

User-Defined Rules
User-defined rules include different heuristics used by the architecture designer to describe characteristics of the envisioned architecture. Some of them translate directly into the settings to 1 or 0 of some cells of the connection matrices. In the example described in section 3, the fact that "DMU_1 has to send his information to DMU_3" results in setting to 1 the cell H_{13}.

Currently the Lattice Algorithm does not allow the taking into account of conditional links of the kind: "if interaction X_{ij} exists then the interaction Y_{jk} exists" or "either link X_{ij} or Y_{jk} exists." An approach is proposed to use techniques from Integer Programming (IP) to translate conditions and logical relations on interactions into constraints that will be checked by the Algorithm. Let us look at two examples illustrating two IP techniques:

a)Either-Or constraints

In a linear program where we want to ensure that at least one of the two constraints $f(x_1, x_2, ..., x_n) \leq 0$ and $g((x_1, x_2, ..., x_n) \leq 0$ is satisfied, we use the following IP formulation:

$f(x_1, x_2, ..., x_n) \leq yM$
$g(x1, x_2, ..., x_n) \leq (1-y)M$

where: $y = 0$ or 1 and M is large enough so that
$f(x_1, x_2, ..., x_n) \leq M$ and $g(x_1, x_2, ..., x_n) \leq M$
for all values of x1, x2, ..., xn.

The statement "either link F_{ij} or H_{ij} exists" is equivalent to the statement "either $1-F_{ij} \leq 0$ or $1-H_{ij} \leq 0$" that can be formulated as:

$1 - F_{ij} \leq y$
$1 - H_{ij} \leq (1-y)$
where $y, F_{ij}, H_{ij} \in \{0,1\}$. (M=1 is large enough).

By adding these two inequalities, we can eliminate y and we get:

$F_{ij} + H_{ij} \geq 1$

It is impossible to have both $F_{ij} = 0$ and $H_{ij} = 0$. This type of constraint can be added to the set of constraints checked by the Lattice Algorithm.

b) If-Then constraints

When a situation occurs where we want to ensure that if a constraint $f(x_1, x_2, ..., x_n) > 0$ is satisfied then the constraint $g(x_1, x_2, ..., x_n) \geq 0$ is satisfied, the following formulation in IP is used:

$- g(x_1, x_2, ..., x_n) \leq My$
$f(x_1, x_2, ..., x_n) \leq M(1 - y)$

where: $y = 0$ or 1 and M is large enough so that
$f(x_1, x_2, ..., x_n) \leq M$ and $- g((x_1, x_2, ..., x_n) \leq M$
for all values of x1, x2, ..., xn.

The statement "If interaction F_{ij} exists then interaction H_{kl} exists" is equivalent to the statement "if $F_{ij} > 0$ then $H_{kl} - 1 \geq 0$" that can be formulated:

$-(H_{kl} - 1) \leq y$
$F_{ij} \leq (1 - y)$
where $y, F_{ij}, H_{kl} \in \{0,1\}$. (M=1 is large enough).

By adding these two inequalities we can eliminate y and we get:

$F_{ij} - H_{kl} \leq 0$ where $F_{ij}, H_{kl} \in \{0,1\}$.

The interpretation is as follows:
If $F_{ij} = 0$ then $H_{kl} = 0$ or $H_{kl} = 1$
If $F_{ij} = 1$ then $H_{kl} = 1$

which is exactly what is required.

Let us generalize this approach to more complex constraints. In what follows, capital letters (A, B, ...) denote cells of the connection matrices e, s, F, G, H or K. A indicates that the connection A must exist (A=1). ¬A indicates that the connection A must not exist (A=0). User

defined constraints can then be expressed in terms of a logical formula that uses the operators $\neg$ (not), $\wedge$ (and), $\vee$ (or), $\rightarrow$ (implies / if-then).

These operators have the following properties:

$A \wedge (B \wedge C) = (A \wedge B) \wedge C = A \wedge B \wedge C$	$\wedge$-associativity
$A \vee (B \vee C) = (A \vee B) \vee C = A \vee B \vee C$	$\vee$-associativity
$A \wedge (B \vee C) = (A \wedge B) \vee (A \wedge C)$	$\wedge$-distributivity
$A \vee (B \wedge C) = (A \vee B) \wedge (A \vee C)$	$\vee$-distributivity
$A \rightarrow B = \neg A \vee B$	
$\neg(A \wedge B) = \neg A \vee \neg B$	DeMorgan's Laws
$\neg(A \vee B) = \neg A \wedge \neg B$	

Using these properties, any logical formula can be transformed into a Conjunctive Normal Form where the formula is expressed as the conjunction (operator $\wedge$) of disjunctions (operator $\vee$). For example $(A \vee B) \wedge (\neg A \vee C) \wedge (B \vee \neg D \vee E)$ is a conjunctive normal form because expressions containing only operators $\vee$ are connected with operators $\wedge$. The first step of the approach is therefore to transform the user specified constraint into a Conjunctive Normal Form. Each of the disjunctions of the resulting expression can then be processed independently. A disjunction contains a finite number (m) of positive literals and a finite number (n) of negative literals. Its generic form is:

$$\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n \vee A_{n+1} \vee A_{n+2} \vee \ldots \vee A_{n+m}$$

which, by using $\vee$-associativity, can be rewritten as:

$$(\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_n) \vee (A_{n+1} \vee A_{n+2} \vee \ldots \vee A_{n+m})$$

and by using the first DeMorgan's law:

$$\neg(A_1 \wedge A_2 \wedge \ldots \wedge A_n) \vee (A_{n+1} \vee A_{n+2} \vee \ldots \vee A_{n+m})$$

Finally, this formula can be rewritten:

$$(A_1 \wedge A_2 \wedge \ldots \wedge A_n) \rightarrow (A_{n+1} \vee A_{n+2} \vee \ldots \vee A_{n+m})$$

Any user defined constraints can be expressed in terms of rules of this type.

Let us now use the IP formulation to derive the equation to be included in the set of constraintsd checked by the Lattice Algorithm:

For $A_1 \wedge A_2 \wedge \ldots \wedge A_n$ to be true, every A_i needs to equal 1 and therefore, we must have: $A_1 + A_2 + \ldots + A_n = n$.

Similarly, for $A_{n+1} \vee A_{n+2} \vee \ldots \vee A_{n+m}$ to be true, at least one of the A_{n+j} has to equal 1 and therefore, we must have:

$$A_{n+1} + A_{n+2} + \ldots + A_{n+m} \geq 1$$

To use the "If-Then" IP formulation, these two equations need to be rewritten as:

$$A_1 + A_2 + \ldots + A_n - n+1 > 0$$
$$A_{n+1} + A_{n+2} + \ldots + A_{n+m} - 1 \geq 0$$

and therefore, the formulation is as follows:

$$-A_{n+1} - A_{n+2} - \ldots - A_{n+m} + 1 \leq My$$
$$A_1 + A_2 + \ldots + A_n - n+1 \leq M(1 - y)$$

where: y = 0 or 1 and M is large enough so that

$-A_{n+1} - A_{n+2} - \ldots - A_{n+m} + 1 \leq M$ and $A_1 + A_2 + \ldots + A_n - n+1 \leq M$

for all values of $A_1, \ldots, A_{n+m}$.

M = 1 is large enough and we have:

$$-A_{n+1} - A_{n+2} - \ldots - A_{n+m} + 1 \leq y$$
$$A_1 + A_2 + \ldots + A_n - n+1 \leq 1 - y$$

By adding these two inequalities to eliminate y, we obtain:

$$A_1 + A_2 + \ldots + A_n - n+1 \leq A_{n+1} + A_{n+2} + \ldots + A_{n+m}$$

If $A_1 \wedge A_2 \wedge \ldots \wedge A_n$ is true then Ai = 1 for all i and the equation becomes :

$$1 \leq A_{n+1} + A_{n+2} + \ldots + A_{n+m},$$

requiring that at least one of the $A_{n+j} = 1$.

If $A_1 \wedge A_2 \wedge \ldots \wedge A_n$ is false then there exists an i in {1, ..., n} such that $A_i = 0$ and therefore:

$$A_1 + A_2 + \ldots + A_n - n+1 \leq 0 \leq A_{n+1} + A_{n+2} + \ldots + A_{n+m}$$

and the A_{n+j} can take any values.

The interesting aspect of this approach is that it generates constraints that can be easily checked by the Lattice Algorithm, in the same way it checks constraints R3 and R4.

4.2.2 Deadlocks and Traps

A detailed introduction to the issues involved in this task have been presented in the last progress report together with a brief description of the algorithm for calculating the deadlock and traps for a general Petri Net structure. The algorithm has been finalized during this reporting period and extended for Hierarchical Petri Net structures. As a result Ms. Jin has completed her Master's thesis, and a technical report containing the algorithms and their application will appear soon. A detailed description of the deadlock and trap algorithm is

presented in this report. The extension of this algorithm to Hierarchical Petri Nets will appear in the technical report.

Algorithm

Given a net N = (P, T, I, O), n = | P | , m = | T | , the extended incidence matrix of the net N is C. I_n is an identity matrix with dimension equals to the number of places in the net N. The algorithm starts by constructing the matrix [I_n , C] and evolves to [D_i , R_i]. [I_n , C] will be modified by linear combinations of its rows. Matrix D_i includes place sets of deadlock supports of net N_i, where N_i denotes the net obtained by taking transitions t_1,..., t_i into account in the PPR matrix (already described in the previous report). Matrix R_i represents the PPR matrix that originated from the extended incidence matrix. In short, [D_i , R_i] is the matrix obtained after the i-th iteration in the following algorithm. After the m-th iteration, D_m includes the deadlock support for the net N_m, which is the net N, and R_m shows the PPR matrix of certain place sets in net N. Some redundancy in the statement of the deadlock supports may occur. They will be eliminated in the algorithm.

Deadlock Algorithm

Step 1 Initialization:

Construct the matrix [I_n , C].
Let $D_0 = I_n$; $R_0 = C$; i= 0 ,
Initial Checking:
Check all rows in [D_0 , R_0] to determine if there is a row *r* that has all its R_0 matrix elements R_{rj} (j= 1 to m) be negative or zero, if so, append this whole row *r* to the matrix [D_0 , R_0].

Step 2 Iterations

Repeat for i = 1 to m (m is the number of transitions in net N)

- Determine J = { j | $R_{ji} > o$ } and K = { k | $R_{ki} < 0$}
- For each (j, k) of J × K , add row *j* and row *k* in [D_0 , R_0] element by element. The adding operation of D_{i-1} matrix and R_{i-1} matrix follow D matrix union rules and PPR union rules, respectively; then append the resulting rows to the matrix [D_{i-1}, R_{i-1}].
- Suppress all the rows in J in which elements $R_{ji} > 0$ ($j \in J$).
- Eliminate from [D_i , R_i] the identical rows. Two rows are identical if and only if every element in one row in matrix D_{i-1} has the same value as the corresponding element in another row in D_{i-1}.
- Now we have a new matrix [D_i, R_i].
- Go to the Repeat statement.

Step 3 Check

Check rows in matrix [D_m , R_m]. Find out those rows in R_m in which every element in the row is either negative or zero. The corresponding rows in D_m represent the place sets of deadlock supports. If there are no such rows in R_m, then this net does not have any deadlock supports, which means there are no deadlocks in this net.

Step 4 End of the Algorithm.

If we need to determine minimal deadlocks, then we can eliminate those place sets that have subsets in each iteration in Step 2, and finally all minimal deadlocks can be determined after the end of the algorithm.

Pre-defined Operation Rules
Because the purpose of using Matrix D_i (i=1, ..., m) is to keep track of the places in a place set that could be the deadlock support of the net R_i during each iteration, the union operation on matrix D_i is basically the union operation of Set theory. Therefore, the union operations on elements in Matrix D_i are defined as follows :

1. Operation rules on elements in matrix D_i :

   a. $1 + 1 = 1, 0 + 0 = 0, 1 + 0 = 1$
   b. These rules are commutative.

2. PPR Union Operation rules for matrix R_i :

   The operation on elements in Matrix R_i (i=1,..., m) are used to compare the preset and postset of a place or a place set. Each time a transition is added, the preset and postset relations may change. The operation rules for matrix R_i are used to keep track of the preset and postset relations whenever a new transition is added. These rules are defined as follows.

1.	1 + 1 = 1;	6	(-1) + 0 = -1;
2.	0+ 0 =0;	7	(-2) + 1 = -1;
3	1 + 0 = 1;	8	(-2) + (-1) = -1;
4	(-1) + (-1) =-1;	9	(-2) + 0 =-1;
5	(-1) + 1 = -1;	10	(-2) + (-2) = -1

Interpretation of Results
The final rows in matrix D are the place sets of the deadlock supports. If we need to find all deadlocks including those that have deadlock supports and those that don't, one way to do this is to consider all combinations of deadlock supports. The result should be all the possible deadlocks. Another important task is to find the minimal deadlocks in matrix D. To achieve this, all we need to do is to compare the rows in the final matrix D_m. Those place sets that contain no other subsets are the minimal deadlocks .

Calculating Trap Supports
To calculate the trap supports in an ordinary Petri Net, we need to find out the place sets whose preset is the superset of its postset.

Example: Given a strongly connected free choice net N shown in Figure 21:

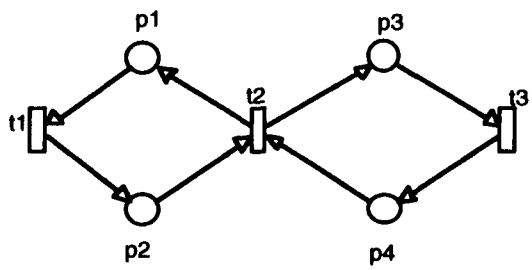

Figure 21 A Strongly Connected FC Net

Follow the algorithm steps described. Each step is shown in two parts as follows: The left hand side shows the matrix iterations, and the right-hand side shows the net N_i that corresponds to i-th iteration. The dashed-line box shows the columns and rows in the current iteration.

Step 1 : Initialization

- Construct matrix $[I_n, C]$. C is the extended incidence matrix of net N, I_n is the identity matrix, $n = |P|$. See Figure 22.
- Let $D_0 = I_n$, $R_0 = C$.
- D_0 represents the deadlock supports of N_0 (with no transitions)
- Check all the rows in $[D_0, R_0]$. There is no row with all its R_0 matrix elements negative or zero. So no rows are appended to the matrix $[D_0, R_0]$.

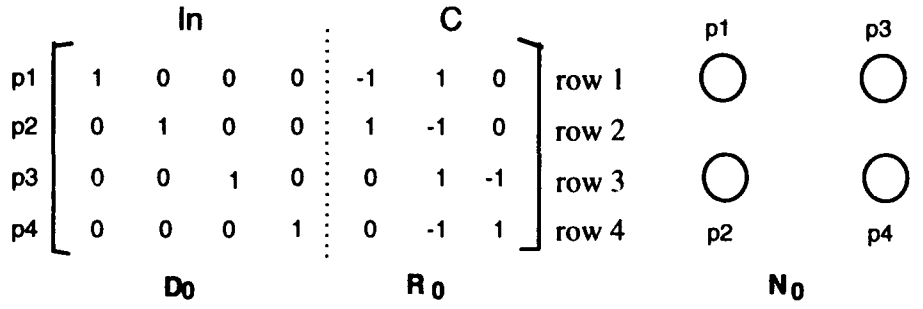

Figure 22 Initialization -- Example

Step 2 : Iterations :

Iteration 1: Add transition t1 to net N_0; the deadlock supports are shown in matrix D_1. R1 shows the preset-postset relation . See Figure 23.

- $J = \{ j \mid R_{j1} > 0 \} = \{2\}$,
- $K = \{ k \mid R_{k1} < 0 \} = \{1\}$,
- $J \times K = \{(2, 1)\}$,
- Add the two rows in $J \times K$ (according to D_i matrix union rules and the R_i PPR union rules), then append the resulting row 5 to the matrix $[D_0, R_0]$.
- Suppress the rows in J, which in this case the only row is 2.
- There are no identical rows in $[D_1, R_1]$, so there are no rows to be eliminated.

  Matrix D_1 contains the deadlock supports of net N1. They are {p1}, {p3}, {p4}, and {p1, p2}. Iteration 1 is summarized as follows :

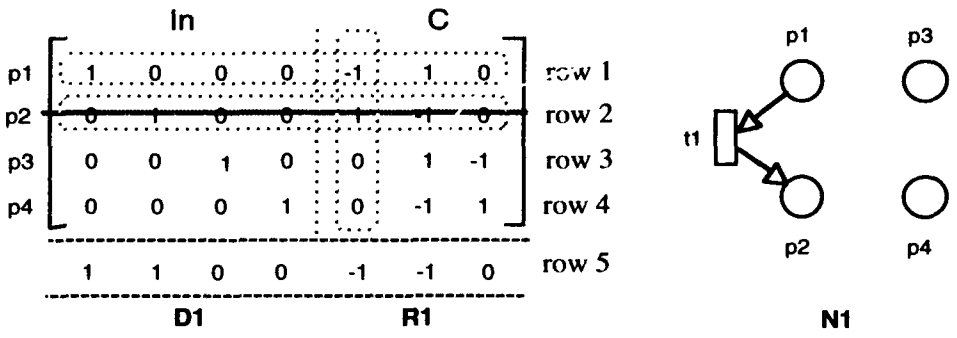

Figure 23 Iteration 1-- Example

Iteration 2: Transition t2 is added to net N_1 to obtain net N2, as show in Figure 24.

- $J = \{ j \mid R_{j2} > 0 \} = \{1, 3\}$,
- $K = \{ k \mid R_{k2} < 0 \} = \{4, 5\}$,
- $J \times K = \{(1, 4), (1, 5), (3, 4), (3, 5)\}$;
- Add rows in $J \times K$ respectively (follow D_i matrix union rules, and R_i PPR union operation rules), append the results as row 6, 7, 8 and 9 to the matrix $[D_1, R_1]$.
- Suppress rows in J, which in this case are row 1 and row 3.
- Row 5 and row 7 are identical, so row 5 is eliminated.
- Now we have $[D_2, R_2]$, D_2 contains the deadlock supports of net N2, they are {p4}, {p1, p4}, {p1, p2}, {p3, p4}, and {p1, p2, p3}. Iteration 2 is summarized as:

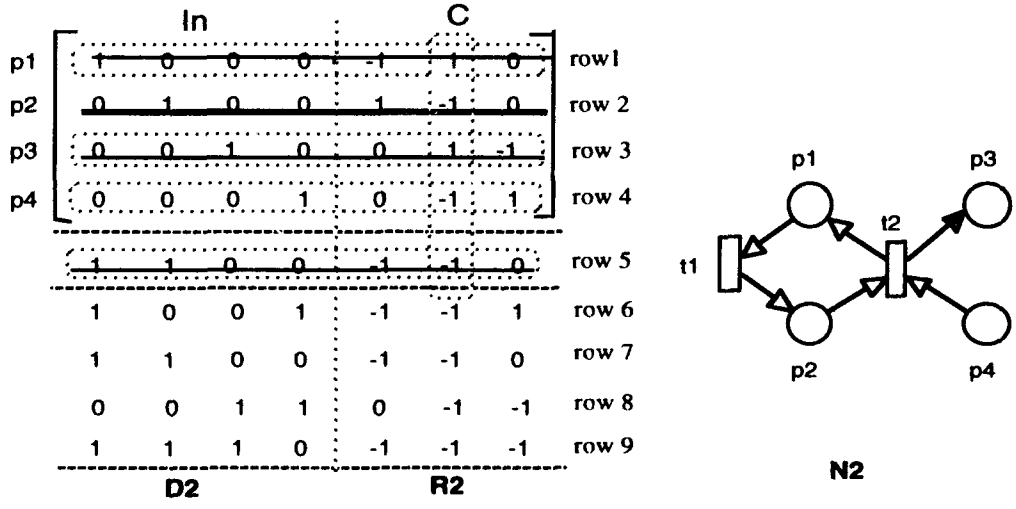

Figure 24 Iteration 2 -- Example

Iteration 3 : Transition t3 is added to the net N2 to obtain net N3. See Figure 25.

- $J = \{ j \mid R_{j3} > 0 \} = \{4, 6,\}$,
- $K = \{ k \mid R_{k3} < 0\} = \{8, 9 \}$,
- $J \times K = \{(4, 8), (4, 9), (6, 8), (6, 9)\}$;
- Add rows in $J \times K$ respectively (according to D_i matrix union rules, and R_i PPR union rules), append the results row 10, row 11, row 12 and row 13 to the matrix $[D_2, R_2]$.
- Suppress rows in J, which in this case is row 4 and row 6.
- Row 13 and row 11 are identical, so are row 8 and row 10, so row 10 and row 11 are eliminated.
- Now we have $[D_3, R_3]$. D_3 contains the deadlock supports of net N3 they are

  {p1, p2}, {p3, p4}, and {p1, p2, p3}, {p1, p3, p4}, and {p1, p2, p3, p4}.

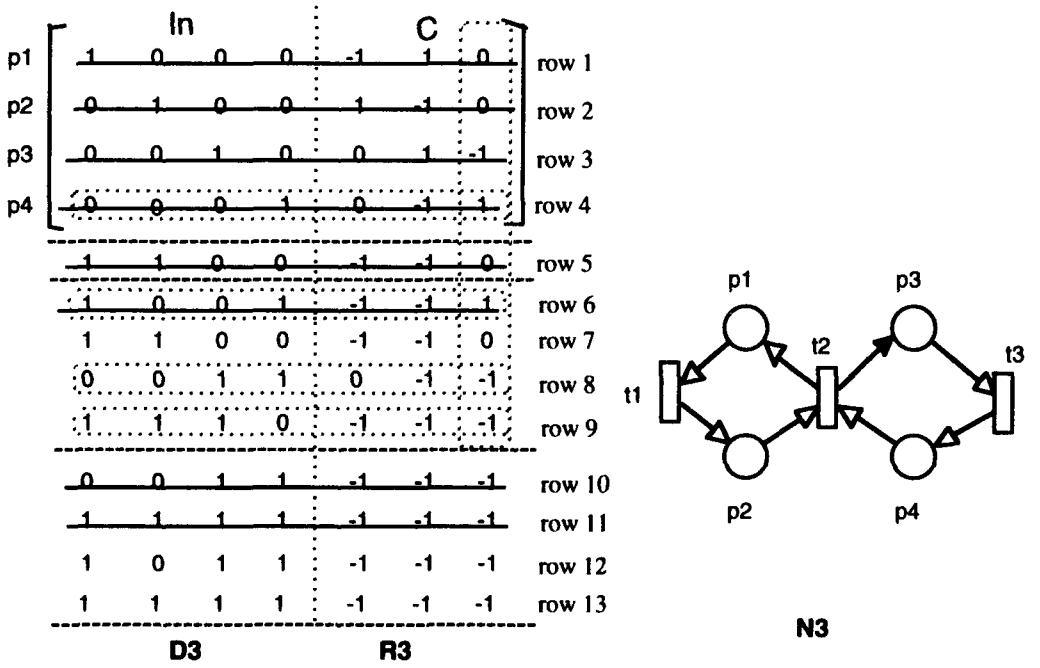

Figure 25 Iteration 3 -- Example

Results : The final matrix of $[D_3, R_3]$ is presented in Figure 26.

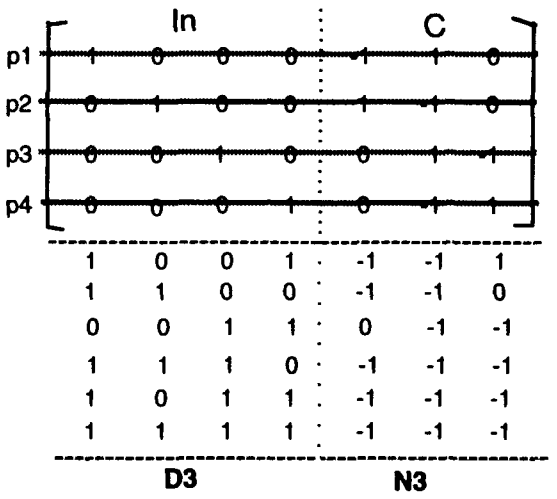

Figure 26 Result -- Example

The final rows in D3 are the followings :

{ 1 1 0 0 } => { p1, p2} ; {0 0 1 1 } => { p3, p4}
{ 1 1 1 0 } => { p1, p2, p3} ; { 1 0 1 1 } => { p1, p3, p4}
{ 1 1 1 1 } => { p1, p2, p3, p4} ;

The minimal deadlocks are:

{p1, p2} and {p3, p4}.

The following are traps and their supports:

{ 1 1 0 0 } => { p1, p2} ; { 0 0 1 1 } => { p3, p4}
{ 1 1 0 1 } => { p1, p2, p4} ; { 0 1 1 1 } => { p2, p3, p4}
{ 1 1 1 1 } => { p1, p2, p3, p4} ;

These place sets are all the trap supports of net N1 of Figure 21. The minimal traps are {p1, p2} and {p3, p4}.

This algorithm is now part of the tool set needed to design and analyze variable structure organization.

4. 3 SUMMARY

The emphasis during the first year of this project has been on fundamental research - the modeling of some abstract problems and the development of algorithms for their solution. With the completion of Ms. Rashba's Master's Project, Ms. Jin's Master's Thesis, and with the results being obtained by Mr. Zaidi for his PhD Thesis, we are now in a position to begin addressing coordination problems in variable structure organizations. The implementation of the results presented in Section 4.2.1 will complete that particular task and will allow us to refocus it on human decision making.

4.4 REFERENCES

Andreadakis S. K. (1988). "Analysis and Synthesis of Decision-Making Organizations." LIDS-TH-1740, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

DeJong G., Mooney R. (1986) "Explanation-Based Learning: An alternative view," Machine Learning 2.

DeJong G., Mooney R. (1990) "Explanation-Based Learning: An alternative view," in Readings in Machine Learning, Shavlik, J. W., Dietterich T. G. (eds), Morgan Kaufman.

Demaël J. (1989). "On the generation of Variable Structure Distributed Architectures." LIDS-TH-1869, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

Hillion H (1986). "Performance Evaluation of Decision Making Organizations Using Timed Petri Nets." LIDS-TH-1590, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

Levis, A. H. (1993). "A Colored Petri Net Model of Command and Control Nodes," in *Toward a Science of Command Control and Communications,* Carl R. Jones, Ed., AIAA Press, Washington, DC.

Lu Z. (1992). "Coordination in Distributed Intelligent Systems." GMU/C3I-120-TH, Center of Excellence in Command, Control, Communications, and Intelligence, George Mason University, Fairfax, VA.

Mitchell T. M., Keller R. M., Kedar S. T. (1986) "Explanation-Based Generalization: A Unifying View," in Machine Learning 1, pp. 47-80.

Mitchell T. M., Keller R. M., Kedar S. T. (1990) "Explanation-Based Generalization: A Unifying View," in Readings in Machine Learning, Shavlik, J. W., Dietterich T. G. (eds), Morgan Kaufman.

Monguillet J.-M. (1987) "Modeling and Evaluation of Variable Structure Organizations." LIDS-TH-1730, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

Remy P. (1986). "On the Generation of Organizational Architectures Using Petri Nets." LIDS-TH-1630, Laboratory for Information and Decision Systems, MIT, Cambridge, MA.

Remy P. and Levis A.H. (1988). "On the Generation of Organizational Architectures Using Petri Nets." in *Advances in Petri Nets 1988, Lecture Notes in Computer Science*, G. Rozenberg Ed. Springer Verlag, Berlin, Germany.

Tecuci G. (1993) "Explanation-Based Learning," Lecture Notes in Machine Learning, Computer Science Dept., George Mason University.

Zaidi S. (1991). "On the Generation of Multilevel Distributed Intelligent Systems Using Petri Nets." GMU/C3I-112-TH, Center of Excellence in Command, Control, Communications, and Intelligence, George Mason University, Fairfax, VA.

5.0 MEETINGS

- Ms. Jenny Jin successfully defended her thesis in April 1994.
- Dr. Levis attended the 1994 Symposium on C2 Research, Monterey, CA.

6.0 CHANGES

No changes in the scope of work of this project.

7.0 RESEARCH PERSONNEL

7.1 Research Personnel — Current Reporting Period

Prof. Alexander H. Levis	Principal Investigator
Mr. Didier Perdu	Graduate Student (Ph.D.)
Mr. Abbas Zaidi	Graduate Res. Assistant (Ph.D.)
Ms. Jenny Jin	Graduate Res. Assistant (MS)

7.2 Research Personnel — Previous Reporting Periods

Prof. Alexander H. Levis	Principal Investigator
Prof. K. C. Chang	
Mr. Didier Perdu	Graduate Student (Ph.D.)
Mr. Abbas Zaidi	Graduate Res. Assistant (Ph.D.)
Ms. Jenny Jin	Graduate Res. Assistant (MS)
Ms. Hedy Rashba	Graduate Res. Assistant
Ms. Azar Sadigh	Graduate Res. Assistant
Ms. Cynthia Johnson	Graphics Designer

7.3 Personnel Changes

Prof. K. C. Chang does not have any current tasking in the project, but is available to consult on algorithmic matters, if the need arises.

Ms. Jin completed her degree requirements and graduated. She is now pursuing her PhD at George Mason University in Software Systems Engineering.

Ms. Rashba completed her degree requirements and graduated. She is now working as an analyst at ANSER Corp., Arlington, VA.

Ms. Cynthia Johnson left the C3I Center to return to industry after only several weeks of employment. She has not been replaced.

8.0 DOCUMENTATION

1. Hedy L. Rashba, "Problems in Concurrency and Coordination in Decision Making Organizations," Report GMU/C3I-143-R, C3I Center, George Mason University, Fairfax, VA, September 1993.

2. Zhenyi Jin, "Deadlock and Trap Analysis in Petri Nets," MS Thesis, Systems Engineering Department, George Mason University, Fairfax, VA, May 1994.